Nutzerfreundliche Modellierung mit hybriden Systemen zur symbolischen Simulation in CLP

Dissertation
zur
Erlangung des akademischen Grades
Doktor-Ingenieur (Dr.-Ing.)
der Fakultät für Informatik und Elektrotechnik
der Universität Rostock

vorgelegt von
Elke Tetzner

Rostock, den 05.09.2008

urn:nbn:de:gbv:28-diss2009-0057-3

Gutachter : Prof. Dr. rer. nat. Dr.-Ing. habil. Günter Riedewald,
Universität Rostock, IEF
Prof. Dr. rer. nat. habil. Michael Hanus,
Christian-Albrechts-Universität zu Kiel, Institut für Informatik
Prof. Dr. rer. nat. habil. Peter Luksch,
Universität Rostock, IEF

Verteidigung: 16.02.2009

Die Dissertation sei all denjenigen gewidmet,
die mich auf dem Weg der Schaffung dieses Werkes begleitet haben und
welchen die Fertigstellung dieser Arbeit eine wahre Freude ist.

Abstract

The PhDthesis for development of the equivalent languages MODEL-HS (MOdular DEclarative Language for Hybrid Systems) and VYSMO (Visual hYbrid Systems MOdelling) is achieved within a framework of former basical works and new works for practical application of the former approaches. As textual and graphical notation MODEL-HS and VYSMO allow the declarative and modular specification of hybrid systems for applications with continuous and discrete components that are described for proving time- and safety-critical properties by the symbolic simulation in CLP (Constraint-Logic Programming) based on formal language theory, which has been developed in Rostock as pioneering and unique approach, specially. In this context, the gap is closed between difficult readability as well as maintainability of complex CLP - descriptions and problem-adequate, user-friendly descriptions by the development of the languages. For the first time, as user-friendly languages on the same formal basis MODEL-HS and VYSMO provide:

1. a declarative, problem-oriented syntax and semantics,
2. an adapted type concept for parametrisation and
3. a modular structure for reusing according to synchronisation and hierarchy

for hybrid systems that serve the symbolic simulation as accepting process of words with time connections. From theory of language, hybrid systems are newly considered under acceptance conditions that can be formally proved and for which many practical examples bear witness. In this way, decidability questions in the area of formal languages can be answered in the future.
Further results serve a broad application of MODEL-HS and VYSMO and should be completed and extended with the view to existing approaches in the future:

- concept of the tool ROSSY (ROStock SYmbolic simulation),
- descriptions of queries and transformation into temporal-logic expressions,
- development of query forms and
- application of MODEL-HS and VYSMO in the areas of study systems and parallel programs.

Keywords : hybrid systems, symbolic simulation, CLP, declarativity, modularity, reusing, synchronisation, hierarchy, acceptance, timed word, languages, modelling

Zusammenfassung

Die vorliegende Dissertation zur Entwicklung der äquivalenten Sprachen MODEL-HS (MOdular DEclarative Language for Hybrid Systems) und VYSMO (Visual hYbrid Systems MOdelling) ist im Rahmen früher, grundlegender und neuer Arbeiten zur praxisnahen Anwendung der frühzeitigen Ansätze entstanden. MODEL-HS als textuelle und VYSMO als graphische Notation erlauben die deklarative und modulare Beschreibung hybrider Systeme zur Modellierung von Anwendungen mit kontinuierlichen und diskreten Komponenten, die speziell dem Nachweis zeit- und sicherheitskritischer Eigenschaften durch die in Rostock als erster und bisher einziger Ansatz entwickelte symbolische Simulation in CLP (Constraint-Logic Programming) basierend auf formalen Sprachen dienen. In diesem Kontext wird durch die Entwicklung der Sprachen die Lücke zwischen der schwierigen Lesbarkeit bzw. Wartung komplexer CLP - Beschreibungen und problemadäquater sowie nutzerfreundlicher Beschreibungen geschlossen. MODEL-HS und VYSMO bieten erstmalig als nutzerfreundliche Sprachen:

1. eine deklarative, problemorientierte Syntax und Semantik,
2. ein angepasstes Typkonzept zur Parametrisierung und
3. eine modulare Struktur zur Wiederverwendung im Sinn von Synchronisation und Hierarchisierung

für hybride Systeme zur symbolischen Simulation als akzeptierender Prozess von Wörtern unter zeitlicher Betrachtung. Aus sprachtheoretischer Sicht wurden hybride Systeme erstmalig unter formal nachweisbaren Akzeptanzbedingungen, die durch praktische Beispiele belegt sind, zur zukünftigen Beantwortung von Entscheidbarkeitsfragen im Bereich formaler Sprachen betrachtet.
Weitere Ergebnisse, die der umfassenden Anwendung von MODEL-HS und VYSMO dienen und zukünftig mit Blick auf vorhandene Ansätze vervollständigt und erweitert werden sollen, ergeben sich aus der:

- Konzeption des Werkzeuges ROSSY (ROStock SYmbolic simulation),
- Beschreibung von Anfragen und Transformation in temporal-logische Ausdrücke,
- Entwicklung von Anfragemasken und
- Untersuchung der Anwendbarkeit von MODEL-HS und VYSMO in Bereichen der Studiensysteme und parallelen Programme.

Schlüsselwörter : Hybride Systeme, Symbolische Simulation, CLP, Deklarativität, Modularität, Wiederverwendung, Synchronisation, Hierarchie, Akzeptanz, Zeitwörter, Sprachen, Modellierung

Inhaltsverzeichnis

Abbildungsverzeichnis

Kapitel 1

Einleitung

Die vorliegende Arbeit widmet sich dem Problem der Entwicklung nutzerfreundlicher Sprachen zur Beschreibung hybrider Systeme, welche durch symbolische Simulation auf der Basis formaler Sprachen in CLP ausgeführt werden. Dabei sind zum einen technische Aspekte wie die Ausführung der symbolischen Simulation und Gestaltung von Anfragen und zum anderen praktische Aspekte wie die Modellierung in unterschiedlichen Anwendungsgebieten, hier Studiensysteme und parallele Programme, als Einflussfaktoren von Entwurfsentscheidungen zu beachten.
Beschreibungen hybrider Systeme im Kontext der symbolischen Simulation müssen gut lesbar, wartbar und wiederverwendbar sein. Die hier entwickelten, deklarativen und modularen Sprachen ermöglichen dem Nutzer eine problemadäquate und überschaubare Modellierung mit hybriden Systemen speziell zur symbolischen Simulation in CLP durch:

- leicht verständliche Elemente und Strukturen der Sprachen mit einem Parameter- und Typkonzept , welche in dieser Arbeit erstmalig für hybride Systeme angepasst auf die symbolische Simulation mit Sicht auf formale Sprachen entwickelt wurden sowie
- Wiederverwendungsmechanismen zur

  Synchronisation und

  Hierarchisierung hybrider Systeme,

  indem ein neues Vorgehen bei der Synchronisation zur Abstraktion und Hierarchisierung durch Verfeinerung erfolgt, welches sich gegenüber bisherigen Vorgehensweisen wie in [Hen00, LSV03, AGLS06] auf eine transitions-basierte Sicht zur symbolischen Simulation als Prozess der Akzeptanz von Wörtern verbunden mit zeitlichen Betrachtungen in hybriden Systemen konzentriert. Der Prozess der Akzeptanz von Wörtern erfordert die Definition eines *wohldefinierten Akzeptanzverhaltens* zur Synchronisation und Hierarchisierung. Bezüglich der *Synchronisation* ist das Akzeptanzverhalten an Ansätze aus [Hen96, BR01, AGLS06] ange-

lehnt. Durch die Betrachtung von Synchronisationsprodukten, welche aus der parallelen Ausführung sich synchronisierender, hybrider Systeme entstehen, wurde eine Einschränkung des modellierten Akzeptanzverhaltens gegenüber einem Produktautomaten ohne Beachtung der Synchronisation erreicht. Aus der Anwendung der transitions-basierten Sichtweise in praktischen Anwendungen zur Modellierung wiederverwendbarer hybrider Systeme wird die Akzeptanz von Wörtern verbunden mit zeitlichen Bedingungen zur *Hierarchisierung* der Systeme neu erklärt.

1.1 Einordnung Hybrider Systeme

Steuerungs- und Überwachungsabläufe administrativer und technischer Bereiche werden heute zum großen Teil durch den Einsatz automatisierter Systeme bestimmt. In zunehmendem Maße sind dabei für die Arbeitsweise der Automatisierungssysteme zeit- und sicherheitskritische Aspekte relevant. Immer mehr Handlungen werden automatisiert, bei denen das menschliche Versagen aufgrund sehr geringer zeitlicher Grenzen zur Ausführung der Handlungen bzw. Gefahrensituationen zu vermeiden ist.
Automatisierungssysteme lassen sich zur Bewältigung technischer Probleme in den Bereich der **Eingebetteten Systeme** (engl. Embedded Systems) [Mar07, RZD$^+$07, Web97] einordnen, da die Softwareanwendungen eng in eine Hardware integriert sind, die durch die Software kontrolliert wird. In den Bereich der **Reaktiven Systeme** (engl. Reactive Systems) [AILS07, Wie03, MP92, MP95], deren wesentliche Aufgabe es ist, eine ständige Wechselwirkung mit der Umgebung der Systeme aufrechtzuerhalten, statt einen Endwert bei der Terminierung zu erzeugen, werden sowohl Automatisierungssysteme zur Lösung administrativer Aufgaben als auch technischer Aufgaben zugeordnet. Da das Verhalten automatisierter Systeme aufgrund des Einsatzes in zeit- und sicherheitskritischen Bereichen vorrangig mit Blick auf einzuhaltende Zeitpunkte und -räume betrachtet werden muss, gehören die Systeme zu den **Echtzeitsystemen** (engl. Real-Time Systems) [CI07, Lap04, BW01]. In Echtzeitsystemen ist die Korrektheit der Ausführung nicht nur von logischen Ergebnissen der Berechnung abhängig, sondern auch von der Zeit, in der die Ergebnisse produziert werden.
In dem beschriebenen Umfeld von Systemen kann der Begriff '**Hybride Systeme**' allgemein entsprechend [ACHH93, ACH$^+$95, vdSS00, LZ05] wie folgt gefasst werden.

Begriff 1.1.1
Hybride Systeme sind Systeme, in denen diskrete und kontinuierliche Komponenten kombiniert werden. Als zeitabhängige Systeme, die mit ihrer Umgebung in ständiger Wechselwirkung stehen, finden hybride Systeme allgemein bei der Steuerung und Überwachung dieser Umgebung Anwendung.

Es handelt sich somit um zeitabhängige, eingebettete, reaktive Systeme, welche durch die Modellierung mit Hilfe diskreter Größen für die Steuerung und Überwachung als auch

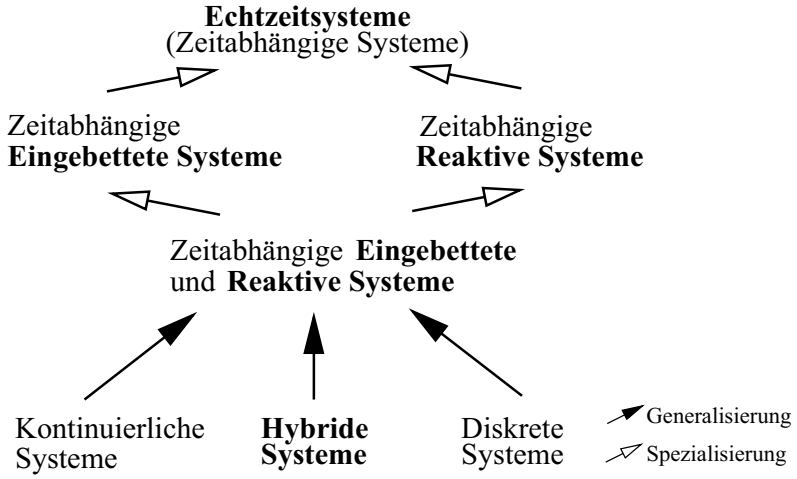


Abbildung 1.1: Klassifikation zu hybriden Systemen

analoger Größen für die sich kontinuierlich ändernde Umwelt gegenüber rein kontinuierlichen bzw. diskreten Beschreibungen besonders anschaulich und realitätsnah dargestellt werden.
In der Abbildung 1.1 ist ein Klassifikationsmodell gegeben, welches die einzelnen Systeme als Klassen betrachtet, deren Hierarchiestufen sich durch Spezialisierung und Generalisierung bilden lassen.
Einsatzgebiete der automatisierten Systeme, wie:

- Transaktionssysteme
- Prozessüberwachung und -steuerung
- Verkehrskontroll- und -leitsysteme
- Kommunikationssysteme
- Studienberatungssysteme,

die mit ökonomischen und lebensbedrohlichen Risiken verbunden sind bzw. Garantien für sachlich und zeitlich geordnete Abfolgen geben müssen [TLR06], erfordern die Prüfung des korrekten Verhaltens der Systeme.

1.2 Symbolische Simulation

Zur Überprüfung von Systemen auf korrekte Verhaltensweisen sind Methoden der Verifikation anzuwenden. Eine ausgewählte Methode ist die symbolische Simulation hybrider Systeme [Rie95, RU95, UR95, Urb96]. Die Methode wurde am Lehrstuhl Programmiersprachen / Übersetzertechnik des IEF der Universität Rostock entwickelt. Allgemein kann eine Klassifikation der symbolischen Simulation mit folgendem Begriff gefasst werden.

Begriff 1.2.1
Die **symbolische Simulation** bildet in der Vorgehensweise eine Art der **klassischen Simulation**, die als Problemlösungsmethode Eigenschaften in Form beispielhafter Experimente eines gegebenen Systems untersucht, und im Erreichen von Ergebnissen eine Art der **Verifikation** , mit welcher ein formaler Nachweis von Eigenschaften in Form vordefinierter Anforderungsspezifikationen für ein gegebenes System geführt wird.

Die Methode der symbolischen Simulation untersucht hier das Verhalten von administrativen und technischen Systemen auf Grundlage formaler Sprachen. *Eigenschaften* dieser Systeme werden über Wörtern spezifiziert, deren Akzeptanz in den für die Systeme modellierten Automaten zu Aussagen in Bezug auf die Korrektheit der Systeme führen. Dabei werden die Eigenschaften nicht nur durch *qualitative Abfolgen* von Ereignissen, sondern unter der Forderung zeitlicher Grenzen für das Auftreten von Ereignissen auch durch *quantitative Annahmen* bestimmt. Die Wörter zur Spezifikation der Eigenschaften müssen somit unter der Berücksichtigung von Zeitvariablen als *Zeitwörter* formuliert werden, wobei jedem Symbol eines Wortes, welches eine Menge von gleichzeitig auftretenden Ereignissen darstellt, eine Variable zur Beschreibung eines symbolischen Zeitpunktes zugeordnet ist.
Die symbolische Simulation hybrider Systeme wird in diesem Ansatz eng mit der Sichtweise der Analyse von Sprachen basierend auf attributierten Grammatiken verbunden, worin ein Unterschied zur Ausführung vergleichbarer Simulationen [NC94, Mis95, MT99, BSW02, Gar02, Tiw02, HW04, LC05, Wan07] besteht. Dadurch kann in der weiteren Forschung ein direkter Zusammenhang zwischen den Ergebnissen der Entscheidbarkeit und Erreichbarkeit, welche auf dem Gebiet der hybriden Systeme errungen werden, mit Entscheidbarkeitsaussagen im Bereich der Sprachen über attributierten Grammatiken geschaffen werden bzw. umgekehrt.
Aus der Modellierung zeit- und sicherheitskritischer Softwareanwendungen auf Grundlage hybrider Automaten [ACHH93, ACH$^+$95, Hen96] ergibt sich, ob und wie spezifizierte Zeitwörter akzeptiert werden. *Endlokationen* in den Automaten lassen Aussagen darüber zu, ob eine spezifizierte Eigenschaft ein Wort der Sprache des hybriden Automaten bildet. *Lokationen* bezeichnen kontinuierliche Komponenten der Automaten, die eine unendliche Menge an konkreten Zustandswerten umfassen. *Bedingungen der Lokationen* und der *Übergänge zwischen den Lokationen* bestimmen, wie die Eigenschaften akzeptiert werden.

1.3 Ausführung in CLP

Die Ausführung der symbolischen Simulation wird in einer constraint-logischen Programmiersprache [JM94, FA97, HW07] vorgenommen, in der hybride Systeme in flachen Strukturen dargestellt sind und Eigenschaften als constraint-logische Anfragen formuliert werden. Constraints bilden dabei Einschränkungen bzw. (Wert-, Rand-, Neben-) Bedingungen, die zur Darstellung unvollständiger Informationen genutzt werden können.

Das Paradigma der constraint-logischen Programmierung (CLP) ist besonders für die Ausführung in unserem Ansatz geeignet, da hier elegant auf hohem Sprachniveau die zwei Paradigmen:

- Logische Programmierung und
- Lösen von Constraints

verbunden werden. Die Programmierungsmethode auf Grundlage einer logik-basierten Regelstruktur ermöglicht die deklarative Beschreibung hybrider Systeme in Form von Fakten und Klauseln, welche während der Ausführung logische Schlussfolgerungen zum Verhalten der Systeme zulässt. Das Lösen über Constraints mit effizienten Methoden:

> vereinfacht die Beschreibung hybrider Systeme und deren Eigenschaften in Form von Zeitwörtern bzw.
>
> führt zum schnellen und erfolgreichen Lösen komplexer Probleme durch die Kombination aus Constraintlösungs- und Suchalgorithmen während der Ausführung.

Auf diese Art und Weise können während der symbolischen Simulation effizient Zusammenhänge zwischen definierten Beziehungen und Bedingungen über Variablen in hybriden Systemen und Zeitwörtern geschaffen werden, die der Beantwortung der Akzeptanz der vorliegenden Zeitwörter in den hybriden Systemen dienen.

1.4 Zielsetzung

Die constraint-logische Programmierung ist nicht zur problemadäquaten Modellierung von Anwendungen durch hybride Systeme geeignet. Dem Sprachparadigma fehlen sowohl syntaktische Strukturen und Elemente zur nutzerfreundlichen Beschreibung hybrider Systeme als auch auf hybride Systeme angepasste, effiziente Ausführungsmechanismen wie konfliktlernende und backtracking-beschleunigende Methoden [FH05] bzw. anwendungsspezifische Vereinfachungen [BLL+06]. In der vorliegenden Arbeit soll deshalb die Lücke zwischen der Beschreibung hybrider Systeme in CLP und der problemadäquaten Modellierung von Anwendungen mit hybriden Systemen durch die Entwicklung nutzerfreundlicher, deklarativer und modularer Sprachen mit spezifischen Strukturen für die Beschreibung und Wiederverwendung hybrider Systeme zur Ausführung der symbolischen Simulation geschlossen werden. Im Wesentlichen sind dazu:

1. Strukturen und Elemente hybrider Automaten zu analysieren und spezifizieren,
2. Strukturen der Wiederverwendung hybrider Systeme und Teile hybrider Systeme zu schaffen,

3. Grundlagen der symbolischen Simulation auf der Basis hybrider Automaten als Prozess der Akzeptanz von Zeitwörtern zu erarbeiten,

4. nutzerfreundliche Sprachen zur Beschreibung hybrider Systeme zu entwerfen und

5. Transformationen neuer Sprachbeschreibungen nach CLP zu ermöglichen.

1.5 Ergebnisse

Im Ergebnis dieser Zielsetzung sind mit MODEL-HS (MOdular Declarative Language for Hybrid Systems) [TR99, TR03, Sik05] als textuelle Notation und VYSMO (Visual hYbrid Systems MOdelling) [Bra00, TBR01, TRB01] als graphische Notation zwei Sprachen zur nutzerfreundlichen Beschreibung von hybriden Systemen für die symbolische Simulation in CLP (Constraint-Logische Programmierung) entstanden. Folgende Merkmale kennzeichnen dabei die Beschreibungen beider Sprachen:

1. gute Lesbarkeit durch problemadäquate Typen, Elemente und Strukturen zur deklarativen Beschreibung hybrider Automaten,

2. Übersichtlichkeit durch die Zusammenfassung hybrider Automaten bezüglich sequentieller Komposition [AGH$^+$00, AG04, AGLS06] und paralleler Komposition [AKY99, LvdBC00] mit Prinzipien der Instantiierung, Modul- und Schnittstellenbildung, Kapselung sowie Umbenennung und Verdeckung,

3. gute Wartbarkeit durch Wiederverwendungsmechanismen bezüglich Abstraktions- und Verfeinerungsprinzipien unter Beachtung der Akzeptanz von Zeitwörtern durch die symbolische Simulation und des zeitlichen Fortschritts in hybriden Systemen sowie

4. Ausführbarkeit der symbolischen Simulation auf unterschiedlichen Abstraktionsebenen.

In Anlehnung an SDL [EHS97, MT01] und kommunizierende Automaten [AKY99, AG04, GOO04] wurden auf der Grundlage sequentieller und paralleler Komposition in [Gor05] einheitliche Abstraktions- und Verfeinerungsmechanismen hybrider Systeme in beiden Sprachen geschaffen, wodurch zukünftig:

- die Überführung der Sprachen ineinander möglich ist,

- MODEL-HS als lineare Notation zu VYSMO betrachtet werden kann und somit

- aus MODEL-HS logische Schlussfolgerungen für VYSMO gezogen werden können.

Folgende Ergebnisse gaben Anregungen zu Verbesserungen und Erweiterungen der Sprachen MODEL-HS und VYSMO:

> [Eic02], wobei ein Rahmenwerk zur Lösung komplexer Fragestellungen entstand, in welchem die symbolische und klassische Simulation in Kombination zur Überprüfung von Eigenschaften in Systemen mit hohem Komplexitätsgrad eingesetzt werden sollen,
>
> [TLR04], worin durch das Werkzeug 'ROSSY' (ROStocker SYmbolic Simulation) eine Architektur zur Modellierung mit hybriden Systemen in MODEL-HS, deren Transformation nach CLP und deren Ausführung durch die symbolische Simulation entwickelt wurde sowie
>
> [LTR03, TR05, TLR06], welche den Einsatz hybrider Systeme im Bereich von Studiensystemen betrachten und [BLL+06, BLLT07a, BLLT07b], wobei hybride Systeme zur Modellierung paralleler Programme genutzt werden.

1.5.1 Weitere Ergebnisse

Ergebnisse, die im Zusammenhang mit der Entwicklung von MODEL-HS und VYSMO entstanden sind, bestehen in:

- Einordnung von MODEL-HS und VYSMO in eine Klassifikation bestehender Sprachen zur Beschreibung hybrider Systeme,
- Untersuchung des Zusammenhangs und der Unterschiede der klassischen Simulation und Verifikation ,
- Einordung der symbolischen Simulation zwischen klassischer Simulation und Verifikation ,
- Schaffung des Zusammenhangs der symbolischen Simulation zum Bounded Model Checking [BCCZ99, BCC+03] , einer Art der automatischen Verifikation mit festgelegten Grenzen für die Ausführung,
- Schaffung einer transitionsbasierten Sichtweise zum Akzeptieren von Zeitwörtern in hybriden Systemen,
- formale Beschreibung hybrider Automaten zur Synchronisation und Bildung von Hierarchien in Bezug auf die symbolische Simulation,
- Transformation hierarchisch hybrider Automaten in flache Strukturen zur Ausführung der symbolischen Simulation in CLP und
- Komplexitätsbetrachtungen zur Transformation in flache Automaten und Ausführung der symbolischen Simulation.

Die Anfragesprache 'MODEL-HS-Query' mit zugehöriger Übersetzung in unsere temporallogische Sprache 'SyS-TPTL' (TPTL for Symbolic Simulation) [Sik06, TSR07], 'VYSMO-Query' [TRB01] sowie die Umsetzung und deren Probleme in nutzerfreundliche Anfragemasken (NUR-Projekt 2005) liefern Ansätze, die weiterführende Ideen für zukünftige Arbeiten enthalten.

1.5.2 Übersicht der Ergebnisse

In der graphischen Abbildung 1.2 sind die Ergebnisse und deren Beziehungen noch einmal in einer Übersicht zusammengefasst. Mit einer Umrahmung sind die drei Schwerpunkte gekennzeichnet, welche grundlegend für die Arbeit sind:

1. nutzerfreundliche Modellierung mit MODEL-HS und VYSMO,
2. Implementation über constraint-logische Programme in CLP und
3. Ausführung der symbolischen Simulation.

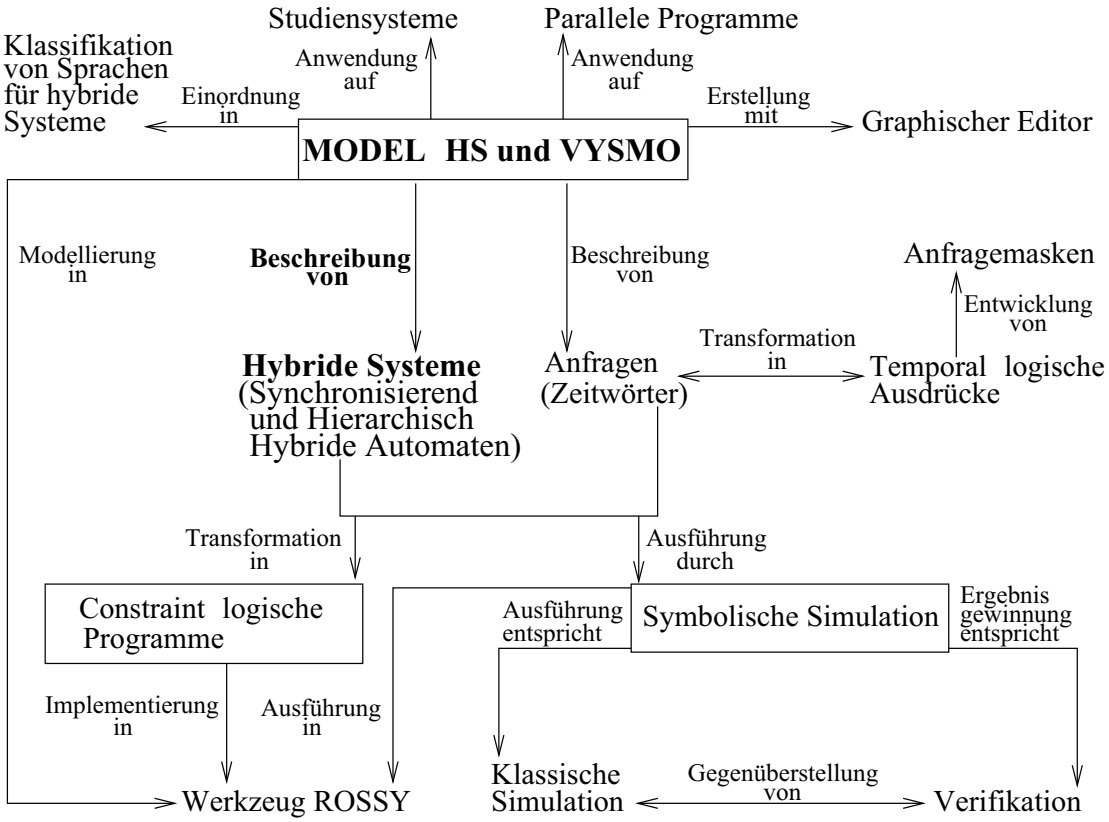


Abbildung 1.2: Ergebnisse im Überblick

Die nutzerfreundliche Modellierung mit *MODEL-HS* und *VYSMO* zur Beschreibung *Hybrider Systeme* auf der Basis *Synchronisierend* und *Hierarchisch Hybrider Automaten* bildet den Kern der Arbeit und ist deshalb vergrößert und fett hervorgehoben. Im Laufe der Entwicklung wurden MODEL-HS und VYSMO mit weiteren Sprachen hybrider Systeme verglichen und in eine aufgestellte *Klassifikation der Sprachen für hybride Systeme* eingeordnet. Die Modellierung in den Bereichen der *Studiensysteme* und der *Parallelen Programme* führten zu neuen Erkenntnissen der Anwendung hybrider Systeme. Zur Erstellung hybrider Systeme mit VYSMO liegt ein *Graphischer Editor* vor. Die nutzerfreundliche Beschreibung von *Anfragen* basierend auf *Zeitwörtern*, deren Hin- und Rücküberführung in *Temporal-logische Ausdrücke* und der Entwicklung nutzerfreundlicher *Anfragemasken*, die mehrdeutige Eingaben zulassen, sollen einen intuitiven Umgang des Ansatzes bezüglich bekannter Methoden, wie temporal-logische Spezifikation des Model Checking [CGP99, HR00, Mer01, BK08] bzw. mehrdeutige Aussagen in der natürlichen Sprache, ermöglichen. Indem hybride Systeme zusammen mit den Anfragen in *Constraint-logische Programme* transformiert werden, ist deren Ausführung durch die *Symbolische Simulation* über Anfragen, welche auf logischer Basis in CLP effizient beantwortet werden können, möglich. Die symbolische Simulation entspricht in der Art und Weise der Ausführung einer *klassischen Simulation* und in der Art der Ergebnisse einer *Verifikation*, wobei die Gegenüberstellung der klassischen Simulation und der Verifikation zu einem Vergleich der Merkmale führt, welche die Bedeutung der symbolischen Simulation in einzelnen Anwendungsbereichen belegt. In der Konzeption des *Werkzeuges ROSSY* wird die Modellierung mit MODEL-HS und VYSMO, die Implementierung in constraint-logischen Programmen und die Ausführung durch die symbolische Simulation verbunden.

1.6 Aufbau der Arbeit

Die Arbeit wurde von einer umfangreichen Literaturrecherche begleitet, weshalb die Beschreibung eng verbundener Ideen und Entwicklungen den folgenden Kapiteln direkt in abgeschlossenen Abschnitten bzw. an vergleichenden und zu erläuternden Stellen zugeordnet ist.
Das Kapitel 2 beschäftigt sich mit der syntaktischen und semantischen Beschreibung hybrider Systeme auf Basis hybrider Automaten in den ursprünglich flachen Strukturen. Im Kapitel 3 werden Sprachen zur Beschreibung hybrider Systeme genauer untersucht und eine Klassifikation bezüglich der formalen Basis, der Notation und des hauptsächlichen Einsatzgebietes vorgenommen. Dabei werden Zusammenhänge zwischen den Sprachen geschaffen, in welche MODEL-HS und VYSMO eingebettet sind. Die genaue Beschreibung modularer Strukturen hierarchisch und synchronisierend hybrider Automaten ist in Kapitel 4 gegeben. Als wesentliche Grundlagen werden die Sprache SDL und kommunizierende Automaten näher betrachtet. Neben einer formalen Beschreibung von hierarchisch hybriden Automaten zur Verfeinerung und synchronisierend hybriden Automaten

zur Abstraktion über Synchronisationsverbindungen unterstützen einführende Beispiele das Verständnis. Verfahren und Werkzeuge zum Nachweis von Eigenschaften bilden das Thema des Kapitels 5. Unter diesem Thema wird eine eingehende Gegenüberstellung von Simulation und Verifikation durchgeführt. Insbesondere werden die Simulation kontinuierlicher, diskreter und hybrider Systeme sowie das Model Checking als eine Art der automatischen Verifikation bezüglich der vorhandenen Literatur untersucht. Auf der Basis von Wörtern und Beschreibungen in CLP ist die symbolische Simulation, welche von der klassischen Simulation und dem Model Checking abgegrenzt wird, in der Ausführung als spezieller Fall des Bounded Model Checking dargestellt. Ein Beispiel für die symbolische Simulation und der aktuelle Stand in der Ausführung runden das Kapitel ab. Die Konzepte von MODEL-HS und VYSMO zur Beschreibung hybrider Systeme werden vergleichend nebeneinander in Kapitel 6 vorgestellt, wobei die Verwendung konzeptioneller Bestandteile und deren Beziehungen durch Beispiele verdeutlicht wird. Ansätze, die im Zusammenhang mit der Entwicklung von MODEL-HS und VYSMO verfolgt wurden, sind im Kapitel 7 überblicksartig skizziert. Eine Zusammenfassung und ein Ausblick auf zukünftige Forschungsrichtungen, die sich aus der nutzerfreundlichen Modellierung mit hybriden Systemen für die symbolische Simulation in CLP und den weiteren Ansätzen aus Kapitel 7 ergeben, schließen die Arbeit im Kapitel 8 ab.
Im Anhang sind Algorithmen und Beispiele enthalten, deren Erarbeitung grundlegend für die erreichten Ergebnissen ist bzw. die Ergebnisse vertiefen. Anhang A und B beinhalten Varianten der Algorithmen zur Beschreibung der Transformation hierarchisch hybrider Automaten in flache Automaten. Im Anhang C ist die Berechnung der transitiven Hülle zur Synchronisation hierarchischer Automaten und deren Komplexität angegeben. Die vollständige Grammatik von MODEL-HS befindet sich im Anhang D. Weitere Beispiele zur Synchronisation und Hierarchiebildung hybrider Automaten sind im Anhang E aufgeführt. Das vollständige Beispiel im Anhang E.3 spiegelt dabei noch einmal einen zusammenhängenden Überblick über Hierarchisierungs- und Synchronisationsmechanismen sowie die Bildung von Zeitwörtern in modular aufgebauten hybriden Systemen wider. Der Anhang F vervollständigt die Beschreibung von Regeln zur symbolischen Simulation und vertieft Zusammenhänge der Berechnung von symbolischen Zeiten und Belegungen von Variablen.

Kapitel 2

Hybride Systeme

Hybride Systeme werden hier als Modelle basierend auf hybriden Automaten [ACHH93, ACH+95, AMP95, AHH96], die in weiteren Kapiteln die Grundlage für die Entwicklung von Automaten in unseren Sprachen VYSMO und MODEL-HS darstellen, eingeführt. Hybride Systeme nach [RU95] auf der Basis hybrider Automaten sind durch folgendes Prinzip gekennzeichnet:

(1) Die vollständige Beschreibung eines hybriden Automaten erfolgt durch Lokationen und Variablen, die von der Zeit abhängig sind. Zu jedem Zeitpunkt befindet sich das System in einer Lokation und weist dort bestimmte Variablenwerte auf. Das Paar aus bestehender Lokation und derzeitigen Variablenbelegungen kennzeichnet somit den Zustand des Systems.

(2) *(Analoges Verhalten)* In jeder Lokation vergeht die Zeit kontinuierlich. Die Variablenbelegungen ändern sich in Abhängigkeit von der Zeit, solange sich das System in einer Lokation befindet.

(3) *(Diskretes Verhalten)* Die Übergänge des Systems von einer Lokation in eine darauffolgende Lokation sind augenblicklich. Diese Übergänge können von Bedingungen abhängig sein und mit einer Zuweisung an die Variablen markiert werden.

In den folgenden Abschnitten werden die syntaktischen Komponenten hybrider Automaten, die Semantik dieser Komponenten und das Verhalten der hybriden Automaten angegeben. Ein Beispiel dient im Anschluss der graphischen Veranschaulichung hybrider Systeme und ihrer Elemente.

2.1 Syntaktische Beschreibung

Die Syntax hybrider Systeme wird definitionsgemäß [RU95, Urb96] eingeführt. Hierbei werden Begriffe und Abkürzungen verwendet, die an dieser Stelle kurz erläutert sind.

Menge der Variablen	:	$x_1, ..., x_m$
Variable für den Zeitraum des Verbleibens in einer Lokation	:	t
Konstanten	:	$k_1, ..., k_m$
$Term$	:	Die Definition eines Terms erfolgt induktiv: 1) Die Variablen $x_1, ..., x_m$ sind Terme. 2) Die Zeitvariable t ist ein Term. 3) Konstanten $k_1, ..., k_m$ sind Terme. 4) Ist F ein n-stelliges Funktionssymbol und sind $m_1, ..., m_n$ Terme, so ist $Fm_1...m_n$ ein Term. 5) Nur die nach 1) bis 4) erzeugten Zeichenreihen sind Terme.
$Gleichung$	:	<$Term$> <$Term$>
$Ungleichung$	:	<$Term$> (< | > | $\leq$ | $\geq$) <$Term$>
$Formel$	:	[$\neg$] (<$Gleichung$> | <$Ungleichung$>) {$\wedge$ <$Formel$>}*

Mit Hilfe dieser Begriffsbildung läßt sich folgende Definition für hybride Systeme angeben, welche auf der Definition aus [RU95] beruht.

Definition 2.1.1
Ein **Hybrider Automat** ist ein Tupel $HA = \langle L, T, l_0, F, Var, Aux, \delta, Act, Inv \rangle$, wobei gilt:

- L ist eine endliche Menge von $Lokationen$.
- T ist eine endliche Menge von $Symbolen$, welche in Anlehnung an SDL [EHS97] als gesendete und empfangene Signale repräsentiert sind. In T existiert ein $faules\ Symbol\ \tau$.
- $l_0 \in L$ ist die $Anfangslokation$.
- $F \subseteq L$ ist eine Menge von $Endlokationen$.
- Var ist eine endliche Menge von $Variablen$.
- Aux ist eine Menge von $binären\ Relationen\ \alpha \subseteq \Gamma(Var) \times \Gamma(Var)$, welche aus einer Menge mit einem Wächter ausgestatteter, nichtdeterministischer Zuweisungen der Form $\psi, a >\!\!\rightarrow\ \{x := [\gamma_x, \eta_x] \,|\, x \in Var\}$ hervorgehen. $\Gamma(Var)$ bezeichnet dabei die Menge der Belegungen aller vorhandenen Variablen. ψ ist eine Formel, a ein Symbol aus T und $\gamma_x,\ \eta_x$ sind zwei Terme.
- δ ist eine $\ddot{U}bergangsrelation$ mit $\delta \subseteq L \times T \times Aux \times L$. Für einen Übergang (l, a, α, l') schreibt man kurz e. Für jede Lokation l gibt es einen $faulen\ \ddot{U}bergang$ der Form (l, τ, Id, l), wobei Id eine identische (antisymmetrische) Relation darstellt.

- Act ist eine Funktion, die jeder Lokation $l \in L$ eine Menge von $Aktivitäten$ zuweist, welche in konjunktiver Beziehung zueinander stehen. Eine $Aktivität$ wird in Form einer Funktion $x \# f(t)$ angegeben, mit $x \in Var$, t ist eine Zeitvariable und $f(t)$ ist ein Term in Abhängigkeit von der Zeit und $\# \in \{<, \leq, =, \geq, >\}$.

- Inv ist eine Funktion, die jeder Lokation $l \in L$ eine $Invariante$ zuweist, welche eine Formel ψ ist.

2.2 Semantische Beschreibung

Die Semantik einzelner Komponenten des hybriden Automaten, die diskrete und analoge Arbeitsweise solch eines Automaten und die Definition eines Laufes ρ, der die Verhaltensweise hybrider Systeme widerspiegelt, sind hier näher betrachtet. Die Komponenten werden folgendermaßen interpretiert:

Die **Symbole** T werden in hybriden Automaten als $Symbole\, zur\, Synchronisation$ zwischen einzelnen Automaten und der Umgebung genutzt. In unseren hybriden Automaten werden Symbole als Menge von Signalen repräsentiert, durch welche die Synchronisation von Automaten wie in SDL [EHS97] geregelt ist. Jedes Signal besitzt einen Status, entweder 'gesendet' oder 'empfangen', welcher die Position der Signale in einer der binären Relationen der Menge Aux bestimmt.

Die **Variablen** Var werden durch eine $Variablenbelegungsfunktion\, \varpi$ in den Bereich der reellen Zahlen $\mathcal{R}$ abgebildet:

$$\varpi : Var \rightarrow \mathcal{R}.$$

Die Menge aller Variablenbelegungen wird mit $\Gamma(Var)$ gekennzeichnet.
Die Zeitvariable t ist ein Element aus dem nichtnegativen Zahlenbereich $\mathcal{R}^+$.
Koeffizienten, die den Variablen zugeordnet werden können, stammen aus dem Bereich der rationalen Zahlen $\mathcal{Q}$.

Die **binären Relationen** Aux werden über der Menge der Variablenbelegungen $\Gamma(Var)$ gebildet, so dass eine Relation $\alpha \subseteq \Gamma(Var) \times \Gamma(Var)$ ist und entsprechend der syntaktischen Regel $\psi, a >\!\!\rightarrow \{x := [\gamma_x, \eta_x] \,|\, x \in Var\}$ interpretiert wird:

$$(\varpi, \varpi') \in \alpha \; gdw. \; \varpi(\psi) \wedge \forall x \in Var : \varpi(\gamma_x) \leq \varpi'(x) \leq \varpi(\eta_x).$$

Ein **Übergang** $e = (l, a, \alpha, l')$ ist auf Grundlage der Interpretation binärer Relationen α im Zustand $< l, \varpi >$ als freigegeben zu betrachten, wenn eine Variablenbelegung $\varpi' \in \Gamma(Var)$ existiert und $(\varpi, \varpi') \in \alpha$ gilt. Der Zustand $< l', \varpi' >$ wird als $Übergangsnachfolger$ des Zustandes $< l, \varpi >$ bezeichnet.

Die **Aktivitäten** Act sind Funktionen, die die Menge der nichtnegativen reellen Zahlen $\mathcal{R}^+$ in die Menge der Variablenbelegungen $\Gamma(Var)$ abbilden. Für die Aktivitäten ist gefordert, dass der Funktionsverlauf dieser Aktivitäten, trotz Fortschreiten der Zeit, gleichbleibend ist, d.h die Aktivitäten müssen zeitinvariant sein:
$l \in L$, $f \in Act(l)$ und $t \in \mathcal{R}^+$.
$(f + t)$ ist ebenfalls eine Funktion in l, $(f + t) \in Act(l)$, wobei
$(f + t)(t') = f(t + t')$.

Die **Invarianten** Inv werden auf Teilmengen der Menge der Variablenbelegungen $\Gamma(Var)$ abgebildet, d.h. $Inv \subseteq 2^{\Gamma(Var)}$. Für eine Lokation $l \in L$ und die lineare Formel ψ über $\Gamma(Var)$ gilt:

$$\varpi \in Inv(l) \; gdw. \; \varpi(\psi).$$

Ein **Systemzustand** wird als ein Paar $< l, \varpi >$ angesehen, wobei l eine Lokation aus L ist und ϖ eine Variablenbelegung. Die Menge der Zustände eines hybriden Systems $\mathcal{H}$ wird mit $\Sigma(\mathcal{H})$ gekennzeichnet.

Die **analoge und diskrete Vorgehensweise** eines hybriden Automaten $\mathcal{H}$ kann wie folgt beschrieben werden:

$diskret$: Das diskrete Verhalten von $\mathcal{H}$ wird durch die Übergänge realisiert. Die Übergänge stellen die Ereignisse dar, während der keine Zeitverzögerungen auftreten. Befindet sich $\mathcal{H}$ im Zustand $\langle l, \varpi \rangle$ und ein Übergang $e = (l, a, \alpha, l')$ ist freigegeben, wobei es einen Variablenzustand $(\varpi, \varpi') \in \alpha$ gibt, so geht $\mathcal{H}$ in den Zustand $\langle l', \varpi' \rangle$ über.

$analog$: Befindet sich $\mathcal{H}$ zum Zeitpunkt t in einem Zustand $\langle l, \varpi \rangle$, so wird bei einer Zeitverzögerung $\bigwedge t$ mit Hilfe der Aktivitäten in l ein neue Variablenbelegung ϖ' berechnet. Die Variablenbelegung muß die Invariante $Inv(l)$ erfüllen. Nach $\bigwedge t$ befindet sich $\mathcal{H}$ im Zustand $\langle l, \varpi' \rangle$.

Die Semantik des hybriden Automaten wird durch den Lauf ρ aus [RU95] beschrieben.

Definition 2.2.1
Ein **Lauf** ρ ist eine endliche oder unendliche Folge

$$\rho : \langle l_0, \varpi_0 \rangle \mapsto \overset{t_0}{\underset{f_0}{}} \langle l_1, \varpi_1 \rangle \mapsto \overset{t_1}{\underset{f_1}{}} \langle l_2, \varpi_2 \rangle \mapsto \overset{t_2}{\underset{f_2}{}} ...$$

von Zuständen $\langle l_i, \varpi_i \rangle \in \Sigma(\mathcal{H})$, nichtnegativen reellen Zahlen $t_i \in \mathcal{R}^+$ und Aktivitäten $f_i \in Act(l_i)$, so dass für alle $i \geq 0$ gilt:

(1) **(Initiierung)** l_0 ist die Anfangslokation und $f_i(0) = \varpi_i$.

(2) **(Analoge Fortsetzung)** Für alle $0 \leq t \leq t_i$ gilt $f_i(t) \in Inv(l_i)$.

(3) **(Diskrete Fortsetzung)** Der Zustand $\langle l_{i+1}, \varpi_{i+1} \rangle$ ist der Nachfolger des Zustandes $\langle l_i, \varpi_i' \rangle$, wobei $\varpi_i' = f_i(t_i)$.

Die Abkürzungen besitzen folgende Bedeutung:

$\langle l_i, \varpi_i' \rangle$	:	$zeitlicher\ Nachfolger$ des Zustandes $\langle l_i, \varpi_i \rangle$
$\langle l_{i+1}, \varpi_{i+1} \rangle$	:	$Nachfolgezustand$ von $\langle l_i, \varpi_i \rangle$
$[\mathcal{H}]$	:	$Menge\ aller\ Läufe$ von $\mathcal{H}$.

Als Anmerkung sei darauf hingewiesen, dass jedes zeitabhängige System, wie in [RU95] beschrieben, mit Hilfe eines hybriden Automaten modelliert werden kann, da dieses zeitabhängige System einen Spezialfall der hybriden Systeme darstellt. Die Menge der Uhren C entspricht der Menge der Variablen Var, welche an den Übergängen ihren ursprünglichen Wert beibehalten können oder den Wert 0 zugewiesen bekommen. Die Aktivitäten zeitabhängiger Systeme lassen sich durch eine lineare Funktion der Art $\psi_l^x[\varpi](t) = k_x * t + \varpi(x)$ wiedergeben, wobei ϖ eine Uhrenbelegung, t die Variable der globalen Zeit, k_x ein rationaler Koeffizient und x eine Uhrenvariable ist. Die Invarianten stellen in diesem Fall Bedingungen an die Zeit dar.
In [RU95] werden symbolische Läufe für hybride Systeme eingeführt, die gegenüber konkreten Läufen auf der Grundlage von Regionen definiert sind. Eine **Region** ist eine Menge von Zuständen, die durch eine Lokation l und eine Formel Ψ charakterisiert ist, so dass $< l, \Psi >$ die Menge aller Zustände $< l, \varpi >$ ist, für die ϖ eine gültige Belegung der Formel Ψ darstellt.

Definition 2.2.2
Ein **symbolischer Lauf** ρ_{sym} für einen hybriden Automaten $\mathcal{H}$ ist eine endliche oder unendliche Folge

$$\rho_{sym} :< l_0, \Psi_0 > \mapsto < l_1, \Psi_1 > \mapsto < l_2, \Psi_2 > \mapsto \ldots$$

von Regionen $< l_i, \Psi_i >$, so dass für alle $i \geq 0$ gilt: $< l_{i+1}, \varpi_{i+1} > \in < l_{i+1}, \Psi_{i+1} > \Leftrightarrow \exists < l_i, \varpi_i >, < l_i, \varpi_i' > \in < l_i, \Psi_i >$, so dass $< l_i, \varpi_i' >$ ein zeitlicher Nachfolgezustand von $< l_i, \varpi_i >$ und der Zustand $< l_{i+1}, \varpi_{i+1} >$ ein diskreter Nachfolgezustand von $< l_i, \varpi_i' >$ ist.

Die Definition bringt zum Ausdruck, dass ein symbolischer Lauf ρ_{sym} die Menge aller Läufe der Form

$$< l_0, \varpi_0 > \mapsto^{t_1} < l_1, \varpi_1 > \mapsto^{t_2} < l_2, \varpi_2 > \mapsto^{t_3} \ldots$$

repräsentiert, wobei $< l_i, \varpi_i > \in < l_i, \Psi_i >$ für alle $i \geq 0$ ist. In der Umkehrung gilt, dass jeder konkrete Lauf ρ durch einen symbolischen Lauf repräsentiert werden kann.

2.3 Beispiel

Das Beispiel soll einen Eindruck der Darstellungsmöglichkeit und des Verhaltens eines hybriden Systems vermitteln. Zu diesem Zweck wird der dem hybriden System zugrundeliegende hybride Automat anhand einer graphischen Abbildung beschrieben.
In der Abb. 2.1 ist das Beispiel eines Fahrzeugs, welches aus dem Stand mit einer konstanten Beschleunigung a eine bestimmte Strecke s zurücklegt, gewählt worden. Nach dem Erreichen eines Weges s, der gleich bzw. größer einem vorgegebenen Weg s_m ist, soll das Fahrzeug mit der gleichen negativ gerichteten Beschleunigung a automatisch auf eine Geschwindigkeit v, die zwischen 0 und einer vorgegebenen Geschwindigkeit v_m liegt, abgebremst und danach wieder beschleunigt werden.

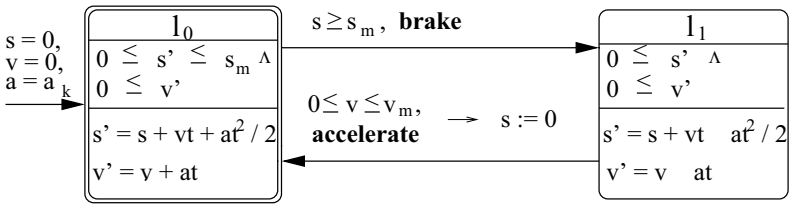


Abbildung 2.1: Automatische Beschleunigung eines Kraftfahrzeuges

Das hybride System ist mit folgenden Komponenten ausgestattet:

Lokationen : l_0, l_1
Symbole : Synchronisation durch $brake$, $accelerate$
Anfangslokation : l_0
Variablen : s, v, a
(s', $v' = Funktionswerte\, f(t)$)
Zeitvariable : t
Parameter(Konstanten) : a_k, s_m, v_m
Binäre Relationen : für $s \geq s_m$, $'brake'$:
$s \geq s_m$ in ϖ_1
für $0 \leq v \leq v_m$, $'accelerate' \rightarrow s := 0$:
$0 \leq v \leq v_m$ in ϖ_2, $s = 0$ in ϖ_2'
Übergangsrelationen : 1) $(l_0, brake, (\varpi_1, \varpi_1'), l_1)$
2) $(l_1, accelerate, (\varpi_2, \varpi_2'), l_0)$
Aktivitäten : in l_0 :
$s' = s + v\,t + a\,t^2/2$ und $v' = v + a\,t$
in l_1 :
$s' = s + v\,t - a\,t^2/2$ und $v' = v - a\,t$
Invarianten : in l_0 :
$0 \leq s' \leq s_m \ \wedge \ 0 \leq v'$

in l_1 :
$0 \leq s' \ \wedge \ 0 \leq v'$

Mit Hilfe der Aktivitäten in den Lokationen lassen sich Fragen wie

- Hat das Fahrzeug zu einem bestimmten Zeitpunkt einen bestimmten Weg zurückgelegt bzw. eine bestimmte Geschwindigkeit erreicht ?
- Wie lang ist der Weg, den das Fahrzeug nach einer erreichten Geschwindigkeit zurückgelegt hat?

beantworten.
Über die Invarianten kann festgestellt werden, ob der Weg und die Geschwindigkeit die Grenzen von 0 bis s_m bzw. v größer 0 eingehalten haben.
An den Übergängen läßt sich prüfen, ob das Signal für das Bremsen zu dem Zeitpunkt erfolgt, in dem sich das System im Zustand l_0 befindet und eine Strecke $s \geq s_m$ zurückgelegt hat, und ob genau dann wieder eine Beschleunigung vorgenommen wird, wenn sich das System im Zustand l_1 befindet und eine Geschwindigkeit v zwischen 0 und v_m erreicht hat.
Der Anfang eines unendlichen, konkreten Laufes ρ für das Verhalten des hybriden Automaten der Abbildung 2.1 sieht bei gegebenen Parametern $s_m = 4$, $v_m = 2$ und $a_k = 2$ zum Beispiel wie folgt aus:
$< l_0, [0,0,2] > \mapsto \overset{2}{\underset{\{s'=2*t^2/2,\, v'=2*t\}}{}} < l_1, [4,4,2] > \mapsto \overset{1}{\underset{\{s'=4+4*t-2*t^2/2,\, v'=4-2*t\}}{}}$
$< l_0, [0,2,2] > \mapsto \overset{1{,}5}{\underset{\{s'=2*t+2*t^2/2,\, v'=2+2*t\}}{}} \ \ldots$.
Die Variablenbelegungen der Lokationen l_0 und l_1 stellen die Werte der Variablen s, v und a beim Eintritt in die jeweilige Lokation dar. Die Übergänge sind durch die Zeiträume des Verbleibens in der vorhergehenden Lokation gekennzeichnet und die ausgeführten Aktivitäten während dieser Zeiträume.
Der zugehörige symbolische Lauf ρ_{sym} besitzt folgende Darstellung:
$< l_0, \ s = 0 \wedge v = 0 \wedge \ a = 2 \wedge 0 \leq s' = a * t^2/2 \leq 4 \wedge 0 \leq v' = a * t > \mapsto$
$< l_1, \ s \geq s_m \wedge 0 \leq s' = s + v * t - a * t^2/2 \wedge 0 \leq v' = v - a * t > \mapsto$
$< l_0, \ 0 \leq v \leq 2 \wedge s = 0 \wedge 0 \leq s' = v * t + a * t^2/2 \leq 4 \wedge 0 \leq v' = v + a * t > \mapsto \ldots$.
Die Formeln der Region zu einer Lokation ergeben sich als Konjunktion der vorausgegangenen Transitionsbedingungen und der Invarianten der betrachteten Lokation. Beide Bedingungen gelten als unbedingte Bedingungen zum Eintritt in die Lokation und weiterem kontinuierlichem Fortschreiten in dieser Lokation.

Kapitel 3

Sprachen zur Beschreibung hybrider Systeme

Veröffentlichungen wie [Cel77] und [Ket94] zeigen, dass Beschreibungen aus einer rein kontinuierlichen Sichtweise schon frühzeitig mit Beschreibungen aus rein diskreter Sicht zur nutzerfreundlichen Modellierung physikalischer Systeme, deren Überwachung und Steuerung automatisch erfolgt, kombiniert wurden. In [Cel77] wird ein kombiniertes System (kombiniertes Modell [ZPK00]) wie folgt definiert:

> Combined systems are systems described, either during the whole period under investigation or during a part of it, by a fixed or variable set of differential equations where at least one state variable or one state derivative is not continuous over a simulation run.

Kombinationen von Beschreibungen wurden folglich aus kontinuierlich beschriebenen Systemen abgeleitet, in denen ein sprunghaftes Verhalten der dynamischen Entwicklung von Zustandsgrößen nachgewiesen werden konnte. Dabei stand die Entwicklung der physikalisch, sich kontinuierlich entwickelnden Systeme im Vordergrund. Je mehr jedoch zeitgleich die automatische Überwachung und Steuerung der Systeme, insbesondere des sprunghaften Verhaltens dieser Systeme, von Interesse war, desto mehr wurden die Aktionen und Reaktionen aus der automatischen Überwachung und Steuerung, beschrieben durch diskret auftretende Ereignisse, als Ausgangspunkte betrachtet. Die diskret auftretenden Ereignisse dienten der Schaffung von diskreten, zustandsbasierten Modellen, die um kontinuierliche Größen erweitert wurden. Ein breiter Überblick verschiedener Richtungen der Beschreibung kombinierter Systeme ist in [vBR00] gegeben.
Beschreibungen kontinuierlich fortschreitender Abläufe kombiniert mit Beschreibungen diskret auftretender Ereignisse wurden unter den Begriffen 'hybride dynamische Systeme' [BGG88, vdSS00] und 'hybride Systeme' [GNRR93] zusammengefasst. Hybride Systeme bilden die Grundlage zur Modellierung von Systemen mit kombiniert kontinuierlichen und diskreten Größen. Diese Art der Modellierung kann laut [vdSS00] in Entwicklungsbereichen wie:

- der Informatik, z.B. zur Verifikation der Korrektheit eingebetteter Systeme unter Echtzeitbedingungen,

- der Steuerungstheorie, z.B. zur Untersuchung von Wechselwirkungsbeziehungen zwischen Datenströmen und physikalischen Prozessen,

- der dynamischen Systeme, z.B. diskontinuierliche Systeme, die schrittweise ausführbare Beispiele für chaotische Systeme liefern können,

- der mathematischen Programmierung, z.B. zur Untersuchung von Optimierungs- und Gleichgewichtsproblemen mit Ungleichungsbedingungen in einem schaltbar dynamischen Rahmenwerk bzw.

- der Simulationssprachen, für welche Bibliotheken sowohl Elemente basierend auf kontinuierlichen als auch diskreten Bestandteilen enthalten,

angewendet werden. Der Begriff 'hybride dynamische Systeme' wird in [vdSS00] wie folgt motiviert:

> Our own background is clearly reflected in the choice of the developments covered, and without doubt the reader will recognize a definite emphasis on aspects that are of interest from the point of view of continuous dynamics and mathematical systems theory. The title that we have chosen is intended to reflect this choice.

In unserem Ansatz wird die Modellierung mittels hybrider Systeme zur Verifikation der Korrektheit eingebetteter Systeme unter Echtzeitbedingungen verwendet, weshalb im Folgenden von dem Begriff 'hybrides System' ausgegangen wird. Allgemeine Modelle zur formalen Beschreibung hybrider Systeme für die Analyse der Systeme sind in [ACHH93] und [ACH$^+$95] entwickelt worden. Der Begriff des hybriden Systems wurde dadurch genau definiert. Mit der Einführung von Sprachen zur Beschreibung hybrider Systeme wurden diese Definitionen bezüglich verschiedener Formalismen konkretisiert. Semantische Beschreibungen der Sprachen basieren dabei auf mathematischen Gleichungssystemen, Logiken, Prozessalgebren, Graphen, Automaten und Petrinetzen. Darüber hinaus wurden hybride Systeme auch speziell über Systemspezifikationen diskreter Ereignisse (DEVS) [ZSKP95, PLPD02], Aktionssystemen [RRS03], abstrakten Zustandsmaschinen (ASM) [Rus05] und I/O Automaten [LSV03] formalisiert. Mit Hilfe problemorientierter syntaktischer Sprachelemente konnte eine nutzerfreundliche Darstellung der hybriden Systeme erreicht werden. Die Handhabung wurde weiterhin durch die Einführung von Strukturen zur Wiederverwendung von hybriden Systemen bezüglich von Verfeinerungs- und Abstraktionsmechanismen verbessert. Im Laufe der Zeit wurden in Anpassung an sich in der Struktur verändernde Anwendungen strukturvariable Mechanismen zur Rekonfiguration von hybriden Systemen geschaffen.

3.1 Ausgewählte Sprachen im Überblick

Die Beschreibung hybrider Systeme dient unterschiedlichen Analysezwecken. Während in den ersten Entwicklungsjahren die Simulation die bestimmendere Art der Analyse darstellte, verstärkte sich mit erhöhten Rechenkapazitäten und -geschwindigkeiten das Interesse an der Verifikation zum korrekten Nachweis von Eigenschaften. Durch einige Sprachen wird die Anwendung beider Analysearten unterstützt. Weiterhin kann ein Trend von der Entwicklung textueller Sprachen zu graphischen Sprachen verzeichnet werden. Vorhandene Rechenkapazitäten, Darstellungsmöglichkeiten und Rechengeschwindigkeiten bilden bei der Wahl der jeweiligen Sprachbeschreibung ausschlaggebende Punkte. Nutzer sollen in der Lage sein, Modelle schnell und leicht in übersichtlicher Art und Weise zu erstellen. Textuelle Beschreibungen können neben graphischen Beschreibungen als äquivalente lineare Notationen bestehen, aus denen logische Schlussfolgerungen für die graphische Notation gezogen werden können.
In der Abbildung 3.1 wurde eine Einteilung ausgewählter Sprachen entsprechend konkreter Referenzen der Tabelle 3.1 für die Beschreibung hybrider Systeme vorgenommen. Die zeitliche Zuordnung der Sprachen richtet sich dementsprechend nach konkreten Arbeiten, die sich mit der Schaffung und Erweiterung von Sprachen in Bezug auf die Beschreibbarkeit hybrider Systeme beschäftigten. Als Basis der Abbildung wurden Formalismen gewählt, welche den Sprachen zugrundeliegen. Die Dreiteilung der Basisachse ergibt sich aus der Kombination mit den Analysezielen. Die Sprachen des ersten Drittels wurden hauptsächlich zum Zweck der Simulation entwickelt, wogegen die Sprachen des letzten Drittels der Verifikation dienen. Sprachen im mittleren Bereich der Basisachse schließen die Lücke zwischen Simulation und Verifikation. Diese Sprachen unterstützen mit Hilfe verfügbarer Werkzeuge beide Ansätze zur Analyse. Die vertikale Zeitachse ist bezüglich der Darstellungsweise hybrider Systeme unterteilt. Dabei werden Sprachen rein textueller Art, Sprachen auf der Grundlage graphischer Primitiven und Sprachen, die beide Notationen äquivalent zur Verfügung stellen, unterschieden.
Beschreibungen der Sprachen 'Modelica', 'Mosila', 'Hybrid χ' und 'ϕ - Calculus' können aus unterschiedlicher Sicht entwickelt werden. Wie durch die Pfeile gekennzeichnet, können in 'Modelica' und 'Mosila' sowohl Beschreibungen basierend auf mathematischen Gleichungssystemen als auch in Anlehnung an Petrinetze erstellt werden. Für 'Hybrid χ' und 'ϕ - Calculus' wurde die Sicht der hybriden Automaten in den Ansatz der Prozessalgebra integriert. Der gestrichelte Pfeil für 'Hybrid cc' weist darauf hin, dass Beschreibungen hybrider Systeme als Algebra von Prozessen zur Übersetzungszeit in Automaten für 'Hybrid cc' transformiert werden, die eine Variante hybrider Automaten im Sinn von [ACHH93] darstellen. Die Sprachen 'CHARON', 'R-CHARON' und 'Masaccio' vereinigen die Möglichkeit der textuellen Darstellung mit der graphischen Darstellung, wenn die graphische Darstellung auch noch nicht detailliert notiert wurde. Textuelle Sprachen wie 'Hybrid Statext', 'ZimOO' und 'MODEL-HS' besitzen in 'Hybrid Statecharts', 'UML^h' und 'VYSMO' graphisch äquivalente Sprachen, die bereits in den einzelnen Elementen festgelegt sind und semantisch gleichwertige Beschreibungen zu den textuellen Notatio-

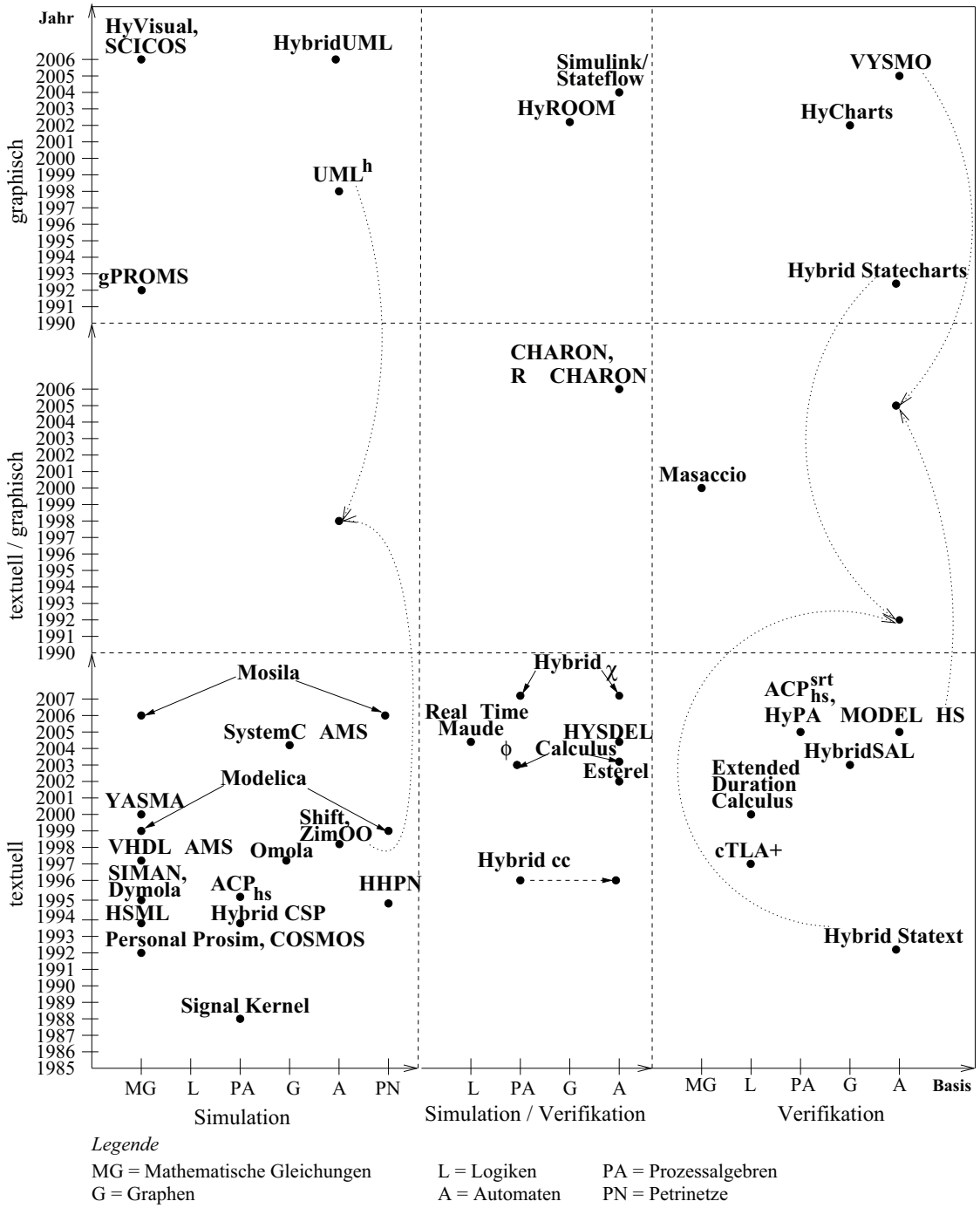


Abbildung 3.1: Entwickelte und Erweiterte Sprachen zur Beschreibung hybrider Systeme

nen liefern sollen. Diese Beziehungen sind mit den gepunkteten Pfeilen in der Abbildung 3.1 gekennzeichnet.

Sprache	*Referenz*	*Sprache*	*Referenz*
ACP$_{hs}$	[Ver95]	**HyROOM**	[BBP$^+$02, SPP01]
ACP$_{hs}^{srt}$	[BM05a]	**HYSDEL**	[TB04, TBB$^+$02]
CHARON	[AGLS06], [AGH$^+$00]	**HyVisual**	[LZ06, LZ05]
		Masaccio	[Hen00]
COSMOS	[Ket92]	**MODEL-HS**	[Sik05, TR99]
cTLA+	[HK97]	**Modelica**	[MOE99, OEM99], [MOE98]
Dymola	[ECO05]		
Esterel	[BS02]	**Mosila**	[ENNG$^+$06]
Extended Duration Calculus	[TH00, ZHL95]	**Omola**	[BM97]
		Personal Prosim	[SdG92]
ϕ - Calculus	[RS03, RS02]	**R - CHARON**	[KSPL06]
gPROMS	[Bar92]	**Real-Time Maude**	[ÖM04]
HHPN	[WS95]	**SCICOS**	[Nik06, NS06], [DLN$^+$99]
HSML	[Tay94, Tay93]		
Hybrid cc	[GJS96, GJSB95]	**Shift**	[DGS98, DGV97]
HyCharts	[GS02, GS98]	**Signal Kernel**	[BGG88]
Hybrid χ	[vBMR$^+$07], [vBMR$^+$06]	**SIMAN**	[PSS95]
		Simulink/Stateflow	[ASK04, Tiw02]
Hybrid CSP	[Jif94]	**SystemC-AMS**	[RG04]
HybridSAL	[Tiw03a, Tiw03b]	**UML^h**	[FNW98]
Hybrid Statecharts	[KP92]	**VHDL-AMS**	[IEE97, DC96]
Hybrid Statext	[KP92]	**VYSMO**	[Gor05, TBR01]
HybridUML	[BBHP06]	**YASMA**	[TF00, FT00]
HyPA	[CR05, MRC05]	**ZimOO**	[FNW98]

Tabelle 3.1: Bibliographie zu den Sprachen

In der Tabelle 3.2 ist eine große Anzahl der Sprachen zur Beschreibung hybrider Systeme unter Gesichtspunkten der Entwicklung zusammengefasst, wobei die einzelnen Sprachbausteine und deren Zusammenwirken durch unterschiedliche Programmierparadigmen und Ziele der Ausführung charakterisiert sind.

Sprache	*Entstehung*	*Art*	*Synchronisation*	*Merkmale*
CHARON	speziell für hybride Systeme	formale Modellierungssprache	1. Version zeitsynchron 2. Version ereignissynchron	agentenorientiert
Dymola	Erweiterung aus	Modellierungs-	ereignis-	objekt-

Tabelle 3.2 siehe nächste Seite

Tabelle 3.2 siehe vorige Seite

Sprache	*Entstehung*	*Art*	*Synchronisation*	*Merkmale*
	Dymola für kontinuierliche Systeme [Elm93]	sprache	synchron über boolsche Variablen	orientiert
Esterel	Erweiterung und Änderung der synchronen Sprache Esterel [Ber00]	Modellierungs-(Programmier-)sprache	synchron	imperativ
Extended Duration Calculus	Erweiterung des Duration Calculus [Zho93]	Modellierungs-(Spezifikations-)sprache	ereignis-synchron	modular
ϕ - Calculus	Erweiterung des π - Kalküls [Mil99]	Spezifikationssprache	asynchron	objekt-orientiert, rekonfigurierbar
Hybrid cc	Erweiterung und Änderung von cc [SJG95] bzgl. synchroner Sprachen	Modellierungs-(Programmier-)sprache	synchron	modular, deklarativ
HyCharts	Erweiterung von ROOMCharts [SGW94]	formale Modellierungssprache	zeitsynchron	modular mit sequentieller und paralleler Komposition
Hybrid χ	speziell für hybride Systeme	Modellierungssprache mit Simulations- und Verifikationswerkzeug	synchron	modular
HybridSAL	Erweiterung von SAL für diskrete Transitionen [BGL+00]	Modellierungssprache	synchron, asynchron	modular
Hybrid-Statext, Hybrid StateCharts	Erweiterung von Statecharts [Har87]	Modellierungssprachen	zeitsynchron	modular
HybridUML	Erweiterung von	Entwurfs-	asynchron	agenten-

Tabelle 3.2 siehe nächste Seite

Tabelle 3.2 siehe vorige Seite

Sprache	*Entstehung*	*Art*	*Synchronisation*	*Merkmale*
	UML Profile 2.0	sprache		orientiert
HyPA	Erweiterung der Syntax der Prozessalgebra ACP [BW90]	Modellierungs-sprache	synchron, asynchron	modular
HyROOM	Erweiterung von ROOM [SGW94]	Spezifikations-sprache	zeitsynchron, ereignis-asynchron	modular
HYSDEL	speziell für hybride Systeme	Modellierungs-sprache	-	angelehnt an Syntax von C
HyVisual	speziell für hybride Systeme	domain-spezi-fische Program-miersprache	-	Block-diagramm-struktur
Masaccio	speziell für hybride Systeme	formales Modell	synchron	modular mit seqen-tieller und und paral-leler Kom-position
MODEL-HS, VYSMO	speziell für hybride Systeme	Spezifikations-sprachen	zeitsynchron, ereignis-synchron	modular mit seqen-tieller und und paral-leler Kom-position
Modelica	Erweiterung von Modelica für kon-tinuierliche Sys-teme [EBB+99]	Modellierungs-sprache mit Werkzeu-gen zur Aus-führung und Bibliotheken	synchron	objekt-orientiert
Mosila	Erweiterung von Modelica für hyb-ride Systeme [MOE99]	Modellierungs-sprache mit dynamischen Objekt-strukturen	synchron, asynchron	objekt-orientiert
Omola	speziell für hybride Systeme	Modellierungs-sprache mit Simulator	synchron	objekt-orientiert

Tabelle 3.2 siehe nächste Seite

Tabelle 3.2 siehe vorige Seite

Sprache	*Entstehung*	*Art*	*Synchronisation*	*Merkmale*
		Omsim		
R-CHARON	Erweiterung von CHARON [AGLS06]	formale Modellierungssprache	zeitsynchron ereignissynchron	objektorientiert, reconfigurierbar
Real-Time Maude	Erweiterung von Maude [CDE+02]	Spezifikationssprache und Analysewerkzeug	zeitsynchron	objektorientiert
SCICOS	speziell für hybride dynamische Systeme	Blockdiagrammmodellierer und -simulator	-	Blockdiagrammstruktur
Shift	speziell für hybride Systeme	Modellierungs-(Programmier-)sprache	synchron	modular, reconfigurierbar
Simulink/ Stateflow	Vereinigung von Simulink und Stateflow	Modellierungssprache	synchron	Blockdiagrammstruktur
UML^h, ZimOO	Erweiterung von UML und Object-Z [Smi00]	Entwurfs- und Spezifikationssprachen	synchron	objektorientiert

Tabelle 3.2: Merkmale repräsentativer Sprachen

Entstehung:
Neben Sprachen wie 'Dymola', 'Extended Duration Calculus' bzw. 'Modelica', die um hybride Aspekte erweitert wurden, sind neue Sprachen wie 'CHARON', 'Hybrid χ' bzw. 'Masaccio' entstanden. Erweiterungen wurden wie im Fall von 'Dymola', 'Extended Duration Calculus' und 'Modelica' aus Sprachen zur Beschreibung rein kontinuierlicher Systeme vorgenommen und im Fall von 'Esterel' sowie 'ϕ - Calculus' aus Sprachen zur Beschreibung diskreter Ereignisse. Der 'ϕ - Calculus' ist dabei z.B. aus dem 'π - Calculus' durch das Hinzufügen aktiver Umgebungen, welche sich kontinuierlich über die Zeit entwickeln, hervorgegangen.

Art:
Nach dem Bereich, in welchem die Sprachen entwickelt wurden, und dem Ziel der Ausführung der Beschreibungen lassen sich *Modellierungs-*, *formale Modellierungs-*, *Spezifikations-*, *Entwurfs-* oder *Programmiersprachen* unterscheiden. Sprachen zur Modellierung führen auf das Gebiet der Simulation zurück. Formale Modellierungssprachen besit-

zen dabei eine vollständig formale Beschreibung der Semantik. Spezifikations- und Entwurfssprachen sind Sprachen, die aus softwaretechnischer Sicht den Phasen der Spezifikation und des Entwurfes des Softwarelebenszyklus zuzuordnen sind. Programmiersprachen sind sowohl mit der Simulation als auch der Softwaretechnik verbunden. Die Modellierungssprache 'Shift' besitzt z.B. ein Laufzeitsystem, wodurch Beschreibungen dieser Sprache als eigenständige Programme ausgeführt werden können. Die Sprache 'Esterel' dagegen stellt mehr eine Modellierungs- als eine Programmiersprache dar. Die Reaktivität der Programme ist z.B. abgeschwächt, indem auf 'else' - Verzweigungen verzichtet wurde.

Synchronisation:
Da hybride Systeme ereignis-gesteuerte und zeit-gesteuerte Systeme [KB00] vereinigen, ist auch in der Synchronisation zwischen den Ereignissen und der Zeit zu unterscheiden. Ereignisse lassen sich über Marken und Signale synchronisieren, welche mit den Transitionen verbunden sind. Die Synchronisation der Zeit ist im Takt der Uhren für die kontinuierlichen Vorgänge in den Lokationen vorzunehmen. Beschreibungen einer Sprache können bezüglich beider Steuerungsarten synchron abgearbeitet werden. Die Art der Synchronisation hängt von dem Zweck der Ausführung ab. Soll ein System vollständig, ohne Annahmen über gleichzeitig stattfindende Aktionen und Abläufe simuliert werden, so erfolgt keine Synchronisation. Aktionen und Abläufe werden dann in einer großen Zahl von Experimenten untersucht. Können jedoch Annahmen über die Gleichzeitigkeit von Aktionen und Abläufen getroffen werden, so sind Mechanismen zur Synchronisation nutzbar. Im Bereich der Simulation lässt sich auf diese Art und Weise die Anzahl der Experimente reduzieren. Im Bereich der Verifikation sind solche Voraussetzungen zur Einschränkung der Komplexität oft unumgänglich. In der Praxis erfordert gerade der Bereich automatischer Überwachungs- und Steuerungseinheiten das Auftreten gleichzeitiger Aktionen und Abläufe, die als Eigenschaften der Systeme nachgewiesen werden müssen. Hier findet die Verifikation ein umfangreiches Anwendungsgebiet. Sprachen wie 'CHARON', 'HyPA , 'HybridSAL' und 'Mosila' lassen asynchrones und synchrones Verhalten zu. In einer ersten Version von 'CHARON' wurde von einer asynchronen Kommunikation der Agenten ausgegangen. Doch in einer zweiten Version der Sprache sollte durch die Synchronisation der diskreten Schritte zu jedem Zeitpunkt sichergestellt werden, dass alle nebenläufigen Agenten denselben Wert jeder gemeinsam genutzten Variable wahrnehmen [AGLS06]. In 'HyPA' wird die Synchronisation der kontinuierlichen Abläufe erzwungen, doch die diskreten Aktionen werden in verschränkter (interleaved) Art und Weise ausgeführt, wobei durch einen speziellen Operator zur Kommunikation die Synchronisation ermöglicht werden kann [CR04]. Module in 'HybridSAL' können mit Hilfe eines Operators synchron bzw. asynchron verbunden werden. Für synchron komponierte Module gilt dabei, dass jedes Modul einen diskreten Schritt gleichzeitig ausführt oder im kontinuierlichen Ablauf alle Module Zeit vergehen lassen. Bei der asynchonen Abarbeitung ist es genau einem Modul zu einem Zeitpunkt erlaubt, eine Transition, diskreter oder kontinuierlicher Art, auszuführen [Tiw03b]. 'Mosila' ermöglicht eine synchrone und asynchrone Kommunikation zwischen Sender- und Empfängerobjekten. Zur synchronen Kommuni-

kation muss das Empfängerobjekt sofort auf ein erzeugtes Ereignis des Senderobjektes reagieren. Zur asynchronen Kommunikation können Ereignisse des Senders in einer Warteschlange des Empfängers zur späteren Abarbeitung aufbewahrt werden.

Merkmale:
Die Sprachen sind in der Strukturierung entsprechend des *imperativen*, *deklarativen*, *objekt-orientierten* bzw. *agenten-orientierten* Paradigmas zu unterscheiden. Deklarative Sprachen wie 'Hybrid CC' bieten zusätzlich die Möglichkeit der Unterteilung von Programmeinheiten in Module. Der Begriff der *Blockdiagrammstruktur*, welcher für die Sprachen 'HyVisual', 'SCICOS' und 'Simulink/Stateflow' verwendet wurde, weist auf die Bildung von Hierarchien durch die Nutzung von Blöcken, wie diese für die numerische Simulation kontinuierlicher Systeme angegeben werden, hin. Die Blöcke liegen in Bibliotheken vor und können selbst Basiselemente oder eine Hierarchie weiterer Unterblöcke darstellen. Die Sprache 'HYSDEL' wurde in der Syntax an 'C' angelehnt, doch die Beschreibung diskreter hybrider Automaten (DHA) ist hier nur in einer flachen Struktur möglich. Diskrete hybride Automaten werden für diskrete Zeit entworfen. Zur Beschreibung von komplexen Systemen mit sich dynamisch ändernder Struktur besitzen Sprachen wie 'Mosila', 'R-CHARON' und 'Shift' Elemente und Ausführungsmechanismen zur Rekonfiguration hybrider Systeme.

Ausführliche Betrachtungen zu verschiedenen Sprachen und Formalismen hybrider Systeme und den Austauschformaten 'HSIF' und 'Metropolis' zur Verwendung unterschiedlicher Beschreibungen in ein und derselben Umgebung sind in [CBP+04, CPPSV06, Kri06] enthalten.

3.2 Anwendung und Hierarchisierung

Einige der aufgeführten Sprachen sollen im Hinblick auf deren Anwendung und die Art der Hierarchisierung von Teilsystemen noch einmal genauer betrachtet werden. Insbesondere Sprachen auf der Basis von Graphen und Automaten werden in die Betrachtung einbezogen, da theoretische und praktische Untersuchungen dieser Sprachkategorie ohne große Transformation vergleichbare Ergebnisse in Bezug auf MODEL-HS und VYSMO liefern.

CHARON:
'CHARON' [AGLS06, AGH+00] ist eine der Sprachen, deren Aufbau hybrider Systeme mit dem Aufbau solcher Systeme in 'MODEL-HS' und 'VYSMO' verwandt ist. Die formale Modellierungssprache wurde mit der Sicht auf eine modulare Spezifikation wechselwirkender hybrider Systeme entwickelt, wobei praktisch deren Kontrollfluss und Einflüsse aus der Umwelt auf der theoretischen Basis einer formalen Semantik für zusammengesetzte und verfeinerte hybride Systembeschreibungen untersucht werden sollen. Anhand

von Beispielen wie das Modell eines Tanks mit Füllstandsüberwachung und der Überwachung bzw. Steuerung der Beinbewegung eines Aibo Roboterhundes von Sony wurde die formale Semantik basierend auf dem beobachtbaren Verhalten und der Komposition der Verhaltensweisen von Unterkomponenten dargelegt. Allgemein werden Komponenten in 'CHARON' durch *Agenten* beschrieben. Zur Schaffung komplexer Agenten mit komplexen Verhaltensweisen werden 2 Hierarchieebenen unterschieden:

1. die Architekturhierarchie und
2. die Verhaltenshierarchie.

Dabei wird die Architekturhierarchie durch parallele Komposition atomarer Komponenten (Agenten) geschaffen. Die Verhaltenshierarchie ergibt sich aus der hierarchisch sequentiellen Komposition im Verhalten eines individuellen Agenten, dessen Kontrollfluss durch einen *Modus* beschrieben wird. Agenten kommunizieren mit der Umgebung über gemeinsam genutzte Variablen und entsprechen den Prinzipien:

1. der Instantiierung zur Wiederverwendung und
2. der Verdeckung zur Vermeidung eines gemeinsamen Zugriffes auf Variablen.

Modi stellen hierarchische Zustandsmaschinen zur Beschreibung diskreter Ereignisse verbunden mit Differentialgleichungen und Randbedingungen (Constraints) über Variablen zur Beschreibung kontinuierlicher Vorgänge dar, die auf Untermodi mit den Verbindungen zwischen den Untermodi basieren können. Modi unterstützen die Prinzipien:

1. der Instantiierung zum Einsatz derselben Modusdefinition in unterschiedlichem Kontext,
2. der Bereichsabgrenzung zur lokalen Deklaration von Variablen innerhalb eines Modus mit Standardbereichsregeln der Sichtbarkeit und
3. der Kapselung zur Festlegung von Schnittstellen- und Implementationsbeschreibungen für die Sichtbarkeit- und Zugriffskontrolle nach und von außen.

Die Sprache 'CHARON' erlaubt Ausnahmebehandlungen wie das Zuvorkommen durch z.B. Abbruch in kritischen Situationen, besitzt ein Gedächtnis für abgelaufene Vorgänge (Historien), lässt auf unterschiedlichen Ebenen zusätzliche Einschränkungen durch beschriebene Constraints zu und unterstützt nichtdeterministische Beschreibungen sowie Beschreibungen von Annahmen über die Umgebung für den Ansatz des annahmegarantierten (assume-guarantee) Schlussfolgerns [HMP01, AG04].

R-CHARON:
'R-CHARON' [KSPL06] ist als Erweiterung von 'CHARON' zur Modellierung rekonfigurierbarer Roboter und großer Transportsysteme wie Luftverkehrsüberwachungsanlagen geschaffen worden. Mit 'R-CHARON' erfolgte erstmalig eine vollständig formale Beschreibung der Rekonfiguration hybrider Systeme in Anlehnung an 'Shift', in welcher

die Idee und ein grundlegendes Beispiel entstanden sind. Auch in 'R-CHARON' wird das Verhalten über Modi beschrieben und die Architektur durch Agenten gebildet. Das System stellt hier eine Konfiguration dar, die die Art und Anzahl der im gesamten System existierenden Modi und Agenten zu einem bestimmten Zeitpunkt angibt. Agenten können geschaffen und zerstört werden, wobei sich auch die Kommunikationsverbindungen zwischen den Agenten dynamisch ändern.

HybridUML:
'HybridUML' [BBHP06] ist eine Modellierungssprache, die UML Profile 2.0 zur Spezifikation hybrider Systeme erweitert. Im Wesentlichen dienen HybridUML-Diagramme der Testfallerzeugung im Bereich von Echtzeitsystemen [BFPT06]. Auf der Grundlage einer Transformation in eine 'Hybrid Low Level Language' (HL3) [Bis05] lassen sich die Modelle simulieren. Die Sprache 'HybridUML' basiert formal auf der ersten Version von 'CHARON', wobei das Auftreten von Ereignissen in asynchroner Weise betrachtet wurde. Die formale Grundlage unterstützt eine eindeutige Interpretation, welche formale Schlussfolgerungen über Spezifikationen durch Model Checking bzw. deduktive Beweise ermöglicht sowie Widersprüche bei der Simulation der Modelle vermeidet. Die Sprache 'CHARON' kann durch die Nutzung als formale Grundlage syntaktisch im UML Format repräsentiert werden, so dass UML - Werkzeuge zur Modellierung und Ausführung von 'CHARON' - Beschreibungen verwendet werden können.

HyCharts:
'HyCharts' [GS02, GS98] wurde zur Spezifikation heterogener Systeme, die komplex und zur Umgebung offen sind, entwickelt. Die Systeme sind in überschaubare, eigenständig verwaltbare Teile zerlegbar, über denen individuelle Schlussfolgerungen getroffen werden können. Der Entwurfsprozess hybrider Systeme wird durch den Aufbau aussagekräftiger Berechnungsmodelle mit einer Sammlung von Operatoren über hierarchischen Graphen erleichtert. Mit Hilfe dieser Sprache wurde z.B. die Überwachung des Höhenniveaus (EHC Electronic High Control) eines Fahrwerkes bei pneumatischer Aufhängung (Federung) in Fahrzeugen von BMW modelliert. Aus Sicht der denotationalen Semantik lassen sich die Modelle auch in Anlehnung an Statecharts [Har87] beschreiben. Den hierarchischen Graphen liegen zwei Interpretationen zugrunde. Zum Ersten lässt sich die Aufbaustruktur in 'HyACharts', die hybriden Datenflussgraphen (Architekturgraphen) entsprechen, graphisch repräsentieren. Zum Zweiten kann die Ablaufstruktur zur Ausführung der Modelle mit einer hybriden Maschine durch 'HySCharts' graphisch repräsentiert werden. 'HySCharts' entsprechen dabei hybriden, hierarchischen Zustandsdiagrammen. Zur Berechnung werden Spezifikationen der 'HySCharts' mit hybriden Maschinen modelliert, welche aus:

1. einer Rückkopplungsschleife,
2. einem kombinatorischem (bzw. diskreten) Teil und
3. einem analogen (bzw. kontinuierlichem) Teil

besteht. Mit Hilfe der Rückkopplungsschleife wird der Zustand (bzw. der Speicher) der Maschine modelliert, wodurch sich die Komponente für jeden Betrachtungszeitpunkt *t* an erhaltene Eingabe- und erzeugte Ausgabewerte zu allen Zeitpunkten vor *t* erinnern kann. Der kombinatorische Teil ist mit der Überwachung des analogen Teils verbunden und besitzt keinen Speicher. Ohne Zeitverzug bildet die kombinatorische Komponente die gegenwärtige Eingabe und den Zustand der Rückkopplungskomponente auf den nächsten Zustand in nichtdeterministischer Weise ab. Die analoge Komponente beschreibt das Ein- und Ausgabeverhalten einer Komponente und erweitert die temporale Dimension, solange sich der kombinatorische Teil untätig verhält. Die analoge Komponente kann eine neue Aktivität auswählen, wenn eine diskrete Änderung der Eingabe durch die Umgebung oder den kombinatorischen Teil erfolgt.
In der Kombination von 'HyACharts' und 'HySCharts' wird graphentechnisch zwischen Verknüpfungen der Knoten, welche Teilgraphen darstellen, und Verknüpfungen der Kanten, welche Verbindungswege mit Ein- und Ausgaben der Teilgraphen darstellen, unterschieden. Verknüpfungen der Knoten sind:

- sequentielle Komposition,
- parallele Ausführung und
- Rückkopplungsschleife.

Verknüpfungen der Kanten werden unterteilt in:

- Kopie einer Eingabe auf genau eine Ausgabe,
- Verbindung mehrerer Eingaben zu einer Ausgabe,
- Verzweigung einer Eingabe in mehrere Ausgaben und
- Vertauschen der Eingaben.

HyROOM:
'HyROOM' [BBP+02, SPP01] ist im Sinn der integrierten modellbasierten Softwareentwicklung entstanden. Das Wesen von Systemen mit Sicht auf deren Verhalten unter zeitlichen Aspekten soll mit wachsender Genauigkeit in Form von formalen Modellen beschrieben werden, welche letztendlich die Grundlage zur automatischen Erzeugung lauffähiger Programme bilden. Somit können Merkmale der Programme wie Funktionalität, Struktur, logische und technische Architektur, Daten, Kommunikation, Scheduling und Fehlertoleranz auf formaler Basis untersucht werden. Auf der anderen Seite ermöglichen gegebene CASE - Werkzeuge für die erzeugten Programme die Ausführung von Läufen zur Anforderungsanalyse und die Erzeugung von Testfällen. Als Anwendungsbeispiel wurde eine Kabelspannungsanlage zum Aufwickeln von Kabeln mit unterschiedlicher Dicke auf Rollen genutzt. 'HyROOM'- Modelle wurden zur Ausführung mit dem CASE - Werkzeugprototypen 'MaSiEd', der auf der ROOM - Methode (Real Object-Oriented Method) basiert, analysiert, simuliert und in C++ übersetzt. Da in 'MaSiEd' keine Erzeugung von Testfällen aufgrund der Nutzung von C++ möglich ist, wurden die Modelle

weiterhin zur Nutzung in 'AutoFocus' transformiert, wo Wächter und Zuweisungen in einer funktionalen Sprache spezifiziert sind und die Testfallerzeugung in Constraint - Logischer Programmierung (CLP) ausgeführt wird. Konzeptionell ist die Erzeugung von Testfolgen hier durch die Formalisierung des Testzweckes über existentielle Spezifikationen realisiert. Die Grundlage zur Überprüfung der Spezifikation bildet die Erreichbarkeit bestimmter Zustände bzw. Bedingungen, welche durch 'Bounded Explicit Model Checking' [EP02, BCC+03] bzw. andere Zustandsraumuntersuchungstechniken verifiziert werden kann. Die formale Semantik von 'HyROOM' basiert auf 'HyCharts' und bildet syntaktisch die Erweiterung einer UML - ähnlichen Sprache. 'HyROOMcharts' bilden 'ROOMcharts', die auf dem Statechart - Formalismus entwickelt wurden, erweitert um kontinuierliche Aktivitäten in Zuständen. Die Aktivitäten werden als Blockdiagramme, wie diese im Bereich der numerischen Simulation zum Einsatz kommen, beschrieben. Zusätzliche Speicher sind den Aktivitäten zur Gewährleistung der Übertragung reellwertiger Nachrichtendaten zwischen den Zustandsmaschinen und den Blockdiagrammen hinzugefügt. 'HyROOMcharts' unterstützen wie 'ROOMcharts' Hierarchien kommunizierender, nebenläufiger Komponenten.

Shift:
'Shift' [DGS98, DGV97] war die erste Sprache, die mit dem Hintergrund der Beschreibung eines Netzwerkes dynamisch rekonfigurierbarer, hybrider Automaten entwickelt wurde. Beispiele wie der Partikeltransport in Teilchenbeschleunigern oder die Modellierung von Verkehrsleitsystemen belegen die Notwendigkeit von ständig zu erzeugenden und zu vernichtenden Teilprozessen während der Laufzeit des Gesamtsystems. Einzelne Teilprozesse können modular beschrieben werden, wobei die Module dynamisch zur Laufzeit über eine Ursache - Wirkungs - Synchronisation kommunizieren. Kontinuierliche Zeitphasen wechseln sich dabei mit diskreten Ereignisübergängen ab. Zur Ausführung der hybriden Automaten wurde ein Algorithmus für die Ursache - Wirkungs - Synchronisation implementiert. 'Shift' - Programme werden zur Ausführung in 'C' - Programme übersetzt. Für die Ausführung der Programme wurde ein 'Shift'- eigenes Laufzeitsystem geschaffen. Das ausführbare Programm simuliert mit seinem Lauf den Entwurf, der im 'Shift' - Quellprogramm spezifiziert wurde. In 'Shift' liegt noch keine Hierarchie vor.

Hybrid Statecharts und Hybrid Statext:
Die graphische Sprache 'Hybrid Statecharts' und deren textuelle Repräsentation 'Hybrid Statext' [KP92] wurden zur realistischen, formalen Modellierung reell vorhandener, reaktiver Systeme entworfen. Die Semantik der hybriden Statecharts basiert auf der Semantik von Statecharts erweitert um die Semantik von Zeitgrenzen an Übergangskanten. An dem Beispiel des Katze-Maus-Problems, wobei sich die Frage ergibt: Fängt die Katze die Maus oder wird die Maus das rettende Mauseloch vor der Katze erreichen?, werden die Repräsentationen beider Sprachen gegenübergestellt. Die Modulhierarchie ist mit Hilfe einer kompositionalen, operationalen Semantik beschrieben, die eine parallele Komposition widerspiegelt. Hier sind Verzögerungs-, Zuvorkommens- und Unterbrechungsmechanismen realisiert. Für kontinuierliche Vorgänge und diskrete Ereignisse existieren persi-

stente Datenvariablen. Für Kommunikationskanäle werden flüchtige Variablen, die durch Zeittransitionen zurückgesetzt werden können, genutzt.

UML^h und ZimOO:
'UML^h' als graphische Entwurfsnotation und deren zugeordnete textuelle und formale Spezifikationssprache 'ZimOO' [FNW98] dienen der genauen und vollständigen Spezifikation komplexer hybrider Systeme. Die Phasen des Entwurfes, der Modellspezifikation und der Implementation sollen durch die Wiederverwendung von Teilsystemen unterstützt werden. In diesem Ansatz ermöglicht die Nutzung der objekt-orientierten Softwareentwicklung die Wiederverwendbarkeit von Komponenten. Komplexe energetische Systeme wie das Beispiel eines Dampfkessels stellen das Anwendungsgebiet der beiden Notationen dar. Beschreibungen in 'ZimOO' bilden auf der einen Seite die Basis für Beschreibungen in 'UML^h' und dienen auf der anderen Seite als Zielbeschreibungen der Entwurfsbeschreibungen aus 'UML^h', die weiter mit dem Werkzeug 'Smile' [KFS95] simuliert werden können. 'UML^h' erweitert 'UML' um ein angepasstes Klassenkonzept zum Entwurf hybrider Systeme. Kontinuierliche, diskrete und hybride Klassen werden als Strukturdiagramme in 'UML^h' eingeführt. Ein sehr früher Zustand des Entwurfes hybrider Systeme wird durch die Nutzung abstrakter Klassen erreicht. In 'ZimOO' werden Verhaltensbeschreibungen genauer spezifiziert. Das gesamte Modell wird anschließend in die Simulationssprache des Werkzeuges 'Smile' abgebildet.

Hybrid cc:
'Hybrid cc' [GJS96, GJSB95] wurde als Modellierungs- und Programmiersprache zur Entwicklung kompositionaler, wiederverwendbarer und deklarativer Modelle für Systeme, die aus elektro-mechanischen und rechnergestützten Elementen bestehen, geschaffen. Hierzu wurden insbesondere Produktfamilien von Fotokopierern betrachtet. Weitere Beispiele bilden Modelle von Billiardspielen, Temperaturüberwachungssystemen und das Katze-Maus-Problem. Algebren von Prozessen werden mit den Eigenschaften der Nebenläufigkeit, dem Verdeckungsprinzip, dem Zuvorkommensmechanismus (Preemption) und verschiedenen Beschreibungen der logischen Zeit mit unterschiedlichen Signalen modelliert, wobei die Nebenläufigkeit unter zeitlichen Randbedingungen zur Bezeichnung 'cc' (concurrent constrained) führt. Das constraint-logische Paradigma (CLP) dient als Programmierbasis, wobei Toleranzanalysen mit Hilfe von Constraintlösern ausgeführt werden können. Jede Frage nach der Erfüllbarkeit einer als temporal-logischer Ausdruck formulierten Sicherheits- bzw. Lebendigkeitseigenschaft in einem hybriden System kann in Form einer constraint-logischen Anfrage an ein Programm in 'Hybrid cc' gestellt und beantwortet werden. Die Programme sind in punktuelle Phasen und Intervallphasen, die der Beschreibung diskreter Ereignisse bzw. kontinuierlicher Vorgänge dienen und alternierend abgearbeitet werden, unterteilt. Zusätzliche Kombinatoren ordnen den einzelnen Intervallphasen folgerichtig die zugehörigen punktuellen Phasen zu.

Masaccio:
'Masaccio' [Hen00] wurde als formales Modell zur Beschreibung hybrider, sich dyna-

misch entwickelnder Systeme eingeführt. Anhand eines Beispiels für das Bahnschrankenproblem, bei welchem Züge einen weiteren Verkehrsweg kreuzen können, wird das mathematische Modell erklärt. Das Modell steht als Grundlage der Schaffung verschiedener Spezifikationssprachen zur Verfügung, in welcher Komponenten wie folgt mit Hilfe:

1. paralleler Komposition atomarer Komponenten,
2. sequentieller Komposition atomarer Komponenten,
3. der Umbenennung von Variablen (Daten) und Lokationen (Kontrolle) sowie
4. des Versteckens von Variablen und Lokationen

aufgebaut werden. Bei der parallelen und sequentiellen Komposition ist hier eine beliebige Schachtelung der Komponenten zugelassen. Atomare diskrete Komponenten werden durch Differenzengleichungen und atomare kontinuierliche Komponenten durch Differentialgleichungen widergegeben. Die mathematische Semantik einer Komponente ist durch die Schnittstelle der Komponente und eine Menge von Ausführungen gegeben. Die Schnittstelle bestimmt mögliche Wege des Einsatzes der Komponente durch die Festlegung, wie die Komponente mit anderen Komponenten in Wechselwirkung steht. Die Menge der für eine Komponente definierten Ausführungen legen das mögliche Verhalten als Folgen von Zustandsänderungen ohne Zeitverzug und mit einer zeitlichen Verzögerung in Echtzeit fest.

ϕ - Calculus:
Die Sprache 'ϕ - Calculus' [RS02, RS03] wurde mit dem Hintergrund der Spezifikation der Syntax und Semantik von Programmiersprachen zur Beschreibung mobiler, physikalischer Agenten geschaffen. Die physikalischen Agenten können sich selbst rekonfigurieren. Theoretische Betrachtungen zum Zweck der Untersuchung des Verhaltens der Agenten werden auf der Basis von Prozessalgebren mit dem Bezug auf Bisimulationstechniken [CGP99, Fok00] durchgeführt. Anhand der Organisation im Gepäckaufbewahrungsbereich eines Flughafens wurde die Sprache 'ϕ - Calculus' zur Simulation und Verifikation mobiler Eigenschaften vorgestellt. Die Verbindung von prozessalgebraischen Techniken mit hybriden Automaten führt zu Spezifikationen, die auf einem Paar (E,P) basieren, wobei:

- E eine kontinuierlich fortschreitende Umgebung und
- P einen diskreten Prozess

darstellt. Der Prozess P wird über einen hybriden Prozessausdruck formalisiert, der folgende Komponenten besitzt:

1. π - Aktionen, die nur den Prozessausdruck ändern,
2. Zeitaktionen, die nur die kontinuierliche Umgebung ändern, und
3. e - Aktionen, die sowohl die Umgebung als auch den Prozessausdruck ändern.

Im 'ϕ - Calculus' wird auf die Verwendung von markierten Aktionen, die der Synchronisation der Überwachung von Bestätigungen (Handshaking) dienen, verzichtet und somit eine asynchrone Abarbeitung gewählt. Wiederverwendung und Hierarchien werden durch die Nutzung objekt - orientierter Strukturen erreicht.

Hybridχ :
Zur Modellierung wechselwirkender, paralleler hybrider Systeme in einem einfachen und intuitiven Weg wurde die Sprache ' Hybridχ' [vBMR$^+$07] entwickelt. Die Sprache dient der Beschreibung großer und komplexer Herstellungssysteme in der Prozessindustrie, wie Brauerei, Steuerungen für chemische und biologische Abläufe bzw. integrierte Schaltkreistechnik. In der Sprache sind:

1. verschiedene Klassen von Variablen:
   - diskrete,
   - kontinuierliche,
   - Sprung-
   - Nichtsprung-
   - algebraische Variablen,
2. atomare Anweisungen und Operatoren,
3. verschiedene Wechselwirkungsmechanismen,
4. konsistente Gleichungssemantik,
5. differentiale, algebraische Gleichungen,
6. Prozessterme für die Bereichseinschränkung,
7. Prozessdefinition und -instanziierung sowie
8. verschiedene nutzerfreundliche syntaktische Erweiterungen

realisiert. Sprungvariablen sind dabei Variablen, die während einer Aktion geändert werden. Nichtsprungvariablen sind Variablen, die während einer Aktion nicht geändert werden und explizit angegeben werden müssen. Die Prozesse besitzen folgende Bestandteile:

1. disjunkte Mengen von:
   a) diskreten Variablen,
   b) Nichtsprung-, kontinuierliche Variablen
   c) algebraische Variablen,
2. Transitionen als unbedingte Kanäle,
3. Initialisierungsteil,
4. Rekursionsdefinition.

Vergleichbar zu 'ϕ - Calculus' werden auf der Grundlage eines integrierten Ansatzes von Prozessalgebren und hybriden Automaten Eigenschaften über Simulation und Verifikation überprüft. Durch gemeinsam genutzte (shared) Variablen wird die Kommunikation der Teilprozesse in einem modularen Aufbau über das Handshaking synchronisiert.

HybridSAL:
'HybridSAL' [Tiw03a, Tiw03b] wurde mit der Idee entwickelt, hybride Systeme für die Sprache 'SAL' [BGL+00] zu verwenden. Hierzu werden die hybriden Systeme korrekt in diskrete endliche Zustandssysteme umgesetzt. Das Beispiel eines einfachen Thermostaten und der Beschleunigung eines Fahrzeuges veranschaulichen die Vorgehensweise. Hier können Kompositionen von Komponenten asynchron bzw. synchron mit Hilfe der symbolischen Simulation, die einer Verifikation für die LTL Logik entspricht, untersucht werden. Diskrete Komponenten werden durch Transitionen aus 'SAL', die mit einem Wächter und einer Aktion ausgestattet sind, beschrieben. Die Syntax kontinuierlicher Komponenten entspricht der Syntax diskreter Transitionen, indem eine *kontinuierliche Transition* für einen Modus spezifiziert wird. Der Wächter der kontinuierlichen Transition identifiziert den Modus und die Zustandsinvariante. Die Aktion der kontinuierlichen Transition stellt die Aktivität der kontinuierlichen Dynamik mit Hilfe von Differentialgleichungen dar. Zur Ausführung werden korrekte diskrete Annäherungen der hybriden Systeme modelliert.

Simulink/Stateflow:
Eingebettete Systeme zur Simulation des kontinuierlichen Verhaltens einer physikalischen Umgebung zusammen mit dem diskreten Verhalten einer Überwachungs- und Steuerungseinheit innerhalb der Umgebung werden oft in einer Verbindung aus Matlab's Simulink und dem Werkzeug Stateflow [Tiw02, ASK04] beschrieben. Beispiele wie Tankstandsüberwachungen werden in der Sprache 'MATLAB Simulink/Stateflow' (MSS) modelliert. Simulink wird zur Modellierung der kontinuierlichen Dynamik und Stateflow zur Spezifikation der diskreten Kontrolllogik und des modalen Verhaltens des Systems verwendet. Zur Überdeckung verschiedener Gebiete der Signalverarbeitung bzw. der beliebigen Kombination von kontinuierlicher Dynamik und diskreten Verhaltensweisen besitzt Simulink eine vielfältige Anzahl an Simulinkblöcken. Signale sind dabei Variablen zur Beschreibung synchroner Verhaltensweisen, die einmal geschrieben und mehrfach gelesen werden. Gemeinsam genutzte (shared) Variablen können dagegen mehrfach beschrieben und gelesen werden. Wesentliche Simulink - Blöcke sind zum Beispiel die folgenden Bestandteile:

1. kontinuierliche Zeitblöcke, z.B. Integratoren, Übergangsfunktionen,
2. Mathematische Operatoren z.B. Produkt, Trigonometrische Funktionen,
3. Quellen: Konstanten, In; Ziele: Out,
4. nichtlineare Elemente: Schalter,
5. Stateflow - Diagramme.

Die 'Stateflow' Modellierungssprache basiert auf hierarchischen Zustandsmaschinen mit diskreten Transitionen zwischen den Zuständen und bildet die Steuerungslogik zur Aktivierung einzelner Modi in Simulink Beschreibungen. Simulink/Stateflow - Modelle wurden zur formalen Interpretation und Analyse in hybride Automaten und Modelle von HybridSAL transformiert, um Werkzeuge der Verifikation anwenden zu können.

Hybrid I/O Automata:
'Hybrid I/O Automata' (HIOA) [LSV03] sind in den vorherigen Ausführungen nicht explizit erwähnt worden, da es sich bei diesem Ansatz um ein sehr umfangreiches Rahmenwerk zur Beschreibung hybrider Systeme, die sowohl der Simulation als auch Verifikation zugrunde liegen können, handelt. Das mathematische Modell der HIOA weist grundlegende Gemeinsamkeiten in der Modellierung der Synchronisation und hierarchischen Struktur von MODEL-HS und VYSMO auf. HIOA's sind zur Modellierung aller Komponenten von hybriden Systemen: physikalische Komponenten, Steuereinheiten, Sensoren, Aktoren, Computersoftware, Kommunikationsdiensten und menschlicher Handlungen, die mit dem Rest des Systems wechselwirken, bestimmt. In dem Rahmenwerk sind zum Ausdruck von Systemgrößen keine Gleichungen einer vorgeschriebenen Form bzw. zur Beschreibung diskreter Transitionen keine speziellen Logiksprachen erforderlich. Ein Hauptmerkmal dieses Ansatzes ist die Unterstützung der Dekomposition hybrider Systembeschreibungen. Von wesentlicher Bedeutung ist dabei die Erfassung der Beschreibung des externen Verhaltens der HIOAs, die diskrete und kontinuierliche Wechselbeziehungen des Systems mit der Umgebung umfasst. Die Beschreibungen beinhalten Abstraktionen und parallele Komposition. HIOAs arbeiten auf der Basis von gemeinsam genutzten (shared) Aktionen zur Beschreibung kontinuierlicher und diskreter Vorgänge. Zwischen HIOAs können neben diskreten Informationen auch kontinuierliche Informationen ausgetauscht werden. Dem Tausch kontinuierlicher Informationen dienen externe Variablen und dem Austausch diskreter Informationen externe Aktionen. Zur Vermeidung von 'Zenoness' in zusammengesetzten Hybriden I/O Automaten sind Strategien entwickelt worden, deren Anwendung die Erhaltung der 'Non-Zenoness', auch als 'Receptiveness' bezeichnet, bei der Komposition von Automaten gewährleistet. 'Zeno' Verhalten tritt auf, wenn die Zeit gegen eine Zeitschranke konvergiert, d.h. keine Ereignisse bzw. Aktionen zum Zeitpunkt der Zeitschranke und danach auftreten können, jedoch davor unendlich viele, welche sich mit dem Zeitpunkt ihres Auftretens immer mehr dieser Schranke nähern.

3.3 Zusammenhänge zwischen den Sprachen

In der Abbildung 3.2 sind noch einmal Sprachen abgebildet, die gemeinsame Eigenschaften besitzen. Zur Kennzeichnung der Abhängigkeiten sind drei Arten von Pfeilen gewählt worden.

1. Einfache Pfeile kennzeichnen die Zugehörigkeit von Eigenschaften zu einzelnen Sprachen.

2. Fettgedruckte, gefüllte Pfeile kennzeichnen die Basis zu weiteren Sprachen.

3. Doppelte, nicht gefüllte Pfeile kennzeichen die Zusammengehörigkeit von Sprachen textueller und graphischer Notation.

Die Namen aller Sprachen sind fett hervorgehoben. Eigenschaften sind in einfacher Schrift dargestellt. Prozesse, durch welche einzelne Sprachen in andere Sprachen überführt wurden und somit deren Basis bilden, sind in eckiger Umrahmung dargestellt.

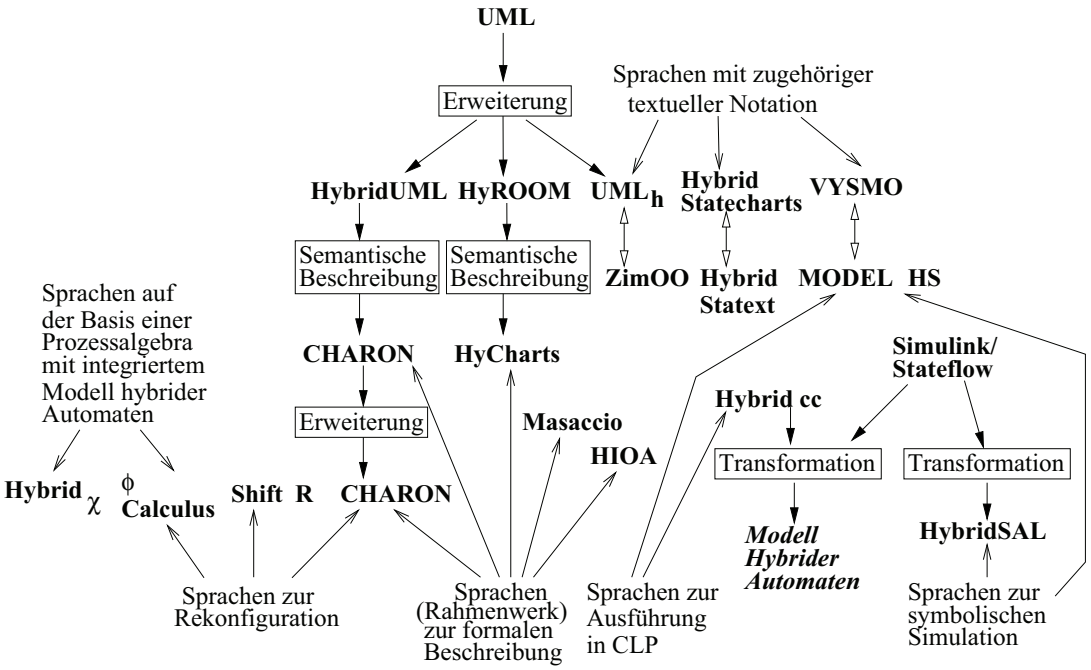


Abbildung 3.2: Abhängigkeiten ausgewählter Sprachen

Die Hybriden I/O Automaten 'HIOA' bilden ein Rahmenwerk, welches der formalen Beschreibung unterschiedlicher Sprachen dient. Auch 'Masaccio' entspricht mehr einem mathmatischen Modell als einer Sprache, da eine bequem handhabbare Syntax fehlt. 'HybridUML', 'HyROOM' und 'UML$_h$' sind als Erweiterungen zu 'UML' entstanden. 'CHARON' bildet als formale Sprache eine semantische Beschreibungsgrundlage für 'HybridUML' und 'HyCharts' eine semantische Beschreibungsgrundlage für 'HyROOM'. Durch die Erweiterung um Strukturen zur Rekonfiguration hybrider Systeme ist aus 'Charon' die Sprache 'R-CHARON' hervorgegangen. Modelle in 'Hybrid cc' und 'Simulink/Stateflow' lassen sich in Modelle auf der Grundlage hybrider Automaten transformieren, welche fett kursiv gekennzeichnet sind. Modelle, die mit Simulink/Stateflow erstellt sind, können weiterhin in Modelle der Sprache 'HybridSAL' transformiert werden.

3.4 MODEL-HS und VYSMO im Vergleich

'MODEL-HS' und 'VYSMO' sind Modellierungssprachen, die in Anlehnung an die Spezifikationssprache SDL [EHS97, MT01] sowie hierarchische und kommunizierende Automaten [AKY99, AGH$^+$00, LvdBC00, AGLS01, AG04, AGLS06] mit dem Hintergrund

der symbolischen Simulation hybrider Systeme, deren Ausführung wie für 'Hybrid cc' in *CLP* erfolgt, vollständig neu entwickelt wurden. In Abbildung 3.3 sind die Eigenschaften dieser Sprachen kurz zusammengefasst.

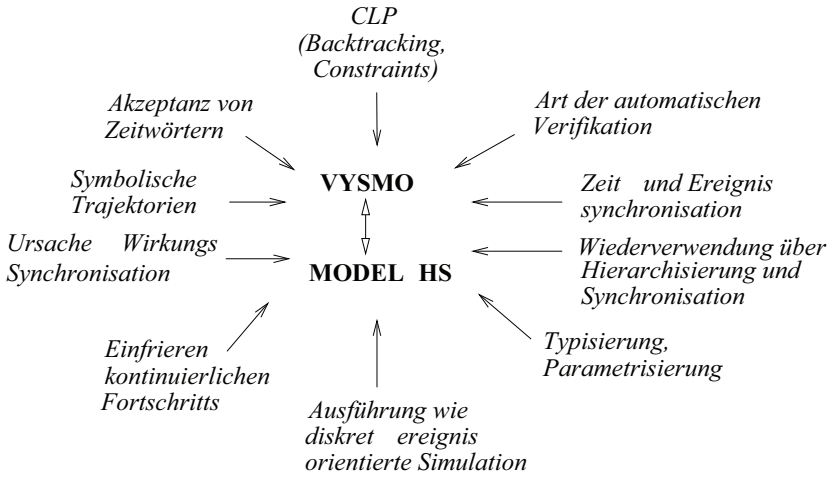


Abbildung 3.3: Eigenschaften von MODEL-HS und VYSMO

Da die symbolische Simulation eine *Art der automatischen Verifikation* darstellt und hauptsächlich Eigenschaften in Systemen mit vorausgesetzten Verhaltensweisen bezüglich der Gleichzeitigkeit auftretender Abläufe und Ereignisse überprüft, kann das *Verhalten* den Automaten sowohl bezüglich der *Zeit* als auch der *auftretenden Ereignisse synchronisiert* werden. Zur Schaffung übersichtlicher und gut lesbarer Programme wurde eine *modulare, deklarative Beschreibungsweise* verwendet. Die Module liegen in Bibliotheken vor und lassen sich zur *Wiederverwendung* über *Hierarchisierung und Synchronisation* als spezielle Arten der sequentiellen und parallelen Komposition wie in 'Masaccio', 'CHARON' und 'HyCharts' verbinden. Auf dieser Basis können schnell erstellbare, leicht überschaubare und wiederverwendbare Beschreibungen zur grundlegenden Untersuchung des Ansatzes der symbolischen Simulation hybrider Systeme geschaffen werden. Dabei sollen schon vorhandene Module, die in Bibliotheken vorliegen, erweiterbar sein.
In Bezug auf die 'HIOA' und 'CHARON' lassen auch 'MODEL-HS' und 'VYSMO' die Deklaration von physikalischen, festgelegten Parametern, Uhren, Variablen und Signalen zu, deren *Typisierung* eng mit der symbolischen Simulation hybrider Systeme verbunden ist. Die Übergabe von Werten bezieht sich hier auf das *Parameterkonzept imperativer Programmiersprachen*. Über Uhren und Variablen können auch kontinuierliche Werte ausgetauscht werden bzw. auf deren kontinuierliches Fortschreiten können mehrere Automaten zugreifen und Einfluss nehmen. Da sich die symbolische Simulation unter Beachtung von Invarianten und Aktivitäten der Lokationen an die *diskret-ereignisorientierte Simulation* anlehnt, wird der kontinuierliche Fluss bei der Ausführung in seinen Randbedingungen betrachtet. Zur Verifikation solcher Eigenschaften wird der kontinuierliche Fluss nicht explizit überprüft, muss jedoch zur Bestimmung der Randbedingungen modelliert werden. Durch die Übergabe von Uhren und kontinuierlichen Variablen als aktuelle Parameter an

formale Parameter der physikalischen, festgelegten Parameter lässt sich das *kontinuierliche Fortschreiten* solcher Werte in Unterautomaten *einfrieren*. Auf diese Art und Weise kann ein unzulässiges Fortschreiten solcher Werte in den Unterautomaten bereits zur Übersetzungszeit erkannt werden.
Wie in 'Shift' wird für unsere Sprachen eine *Ursache-Wirkungs-Synchronisation* mit Hilfe von Signalen modelliert und ausgeführt. Ähnlich zu 'HybridSAL' wird die symbolische Simulation über *symbolische Trajektorien* ausgeführt. In 'HybridSAL' werden wie auch in 'Shift' explizit erschaffene Kellerspeicher eingesetzt. In unserem Ansatz wird durch den Einsatz von CLP das implizit gegebene *Backtrackingverfahren* eingesetzt. Im Gegensatz zur symbolischen Simulation in 'HybridSAL' bilden unsere symbolischen Trajektorien *Zeitwörter*, eine Folge von Ereignissen als Symbole mit der symbolischen Zeit ihres Auftretens, welche in den hybriden Automaten durch die symbolische Simulation auf *Akzeptanz* überprüft werden sollen.

Kapitel 4

Hierarchische und synchronisierende Automaten

Die Sprachen MODEL-HS und VYSMO enthalten Konzepte zu Hierarchisierung und parallelen Komposition mittels Synchronisation in hybriden Systemen. Dabei werden zwei Typen von Automaten unterschieden, der hierarchisch hybride Automat HHA und der synchronisierend hybride Automat SHA , welche in VYSMO direkt umgesetzt sind. In MODEL-HS wurden Automaten und Blöcke nach dem Vorbild von SDL [EHS97, MT01] geschaffen, welche so erweitert sind, dass Automaten den HHAs und Blöcke den SHAs entsprechen. Der hierarchisch hybride Automat HHA wird zur Modellierung einer Verfeinerung, die der sequentiellen Komposition [AGH$^+$00, AG04, AGLS06] entspricht, und der synchronisierend hybride Automat SHA zur Modellierung einer Abstraktion, die der parallelen Komposition [AKY99, LvdBC00] mittels Synchronisation entspricht, eingesetzt. Wie in CHARON werden für hierarchisch hybride Automaten Prinzipien der Instantiierung, Modul- und Schnittstellenbildung sowie Kapselung angewandt und für synchronisierend hybride Automaten das Prinzip der Instantiierung, der Umbenennung und der Verdeckung.
In diesem Abschnitt werden die Automaten in ihren formalen Bestandteilen differenzierter betrachtet, wodurch eine exakte Beschreibung bezüglich der Verfeinerung und Synchronisation möglich ist. Dabei wird gezeigt, dass HHA und SHA selbst hybride Automaten aus der Definition 2.1.1 darstellen. Somit können die beiden Arten von Automaten beliebig ineinander geschachtelt werden. Durch die Schachtelung entsteht ein Gesamtsystem SHHA (Synchronisierend Hierarchisch Hybrider Automat) , das in einen hybriden Automaten der Definition 2.1.1 überführt und als Komponente eines neuen Systems wiederverwendet werden kann. Die Überführung eines SHHA wird anhand der Überführungen der Bestandteile HHA und SHA in flache hybride Automaten gezeigt. Nachdem die Sprache SDL und kommunizierende Automaten im Überblick zum grundlegenden Verständnis eingeführt wurden, erfolgt eine Formalisierung und Transformation der Grundbestandteile HHA und SHA mit detaillierten Beschreibungen zur Verfeinerung und Synchronisation.

4.1 SDL - Specification Description Language

Grundlegende Prinzipien, die der Beschreibung und Wechselwirkung unserer hybriden Systeme zugrunde liegen, sind der Spezifikations- und Beschreibungssprache SDL entnommen worden. SDL wurde ursprünglich als eine Standardspezifikationssprache für das Gebiet der Telekommunikation entwickelt. Konzepte, die in SDL realisiert wurden, sind dementsprechend eng mit der Arbeitsweise von Telekomunikationsanlagen verbunden. In [Pri01] wurde für eine Teilmenge von SDL eine formale Semantik geschaffen, wodurch SDL Notationen zu Analysezwecken von Korrektheit und Vollständigkeit genutzt werden können [GGP03, GG07]. Die Sprache SDL zeichnet sich durch die Verbindung einer textuellen mit einer äquivalenten graphischen Beschreibungsmöglichkeit für komplexe Echtzeitsysteme aus. Auf der Grundlage von Automaten, die sich über ausgesendete und empfangene Signale synchronisieren und über Variablenwerte kommunizieren, werden einzelne Prozesse eines Systems modelliert und simuliert. Die Automaten besitzen dabei klar definierte Schnittstellen zur Umgebung und eine Beschreibung des Verhaltens mit Hilfe von Zuständen und Transitionen. Zur besseren Übersicht können Prozesse des Systems zu Einheiten, sogenannten Blöcken, zusammengefasst werden.

4.1.1 Prinzip des Sendens und Empfangens

Hier wird die Idee der Synchronisation des parallelen Verhaltens von Prozessen bei dem Auftreten gleicher Ereignisse und des unabhängigen Verhaltens voneinander bei verschiedenen Ereignissen genutzt. Aus dem Verhalten einzelner Prozesse und deren Wechselbeziehung soll das Gesamtverhalten des Systems abgeleitet werden. Einige Gründe für die Orientierung an SDL sind:

- In dieser Sprache wird ein System als eine Gesamtheit von kommunizierenden Prozessen betrachtet, wodurch SDL zur Beschreibung paralleler Systeme besonders geeignet ist.

- Die Prozesse in SDL werden über endliche Automaten, deren Zustandsraum durch Variablen erweitert wurde, modelliert. Dieser Ansatz ist mit der Modellierung der Prozesse über hybride Automaten vergleichbar.

- In [FGG+05] wurde die Anwendung von SDL für Echtzeitsysteme gezeigt. Da in unseren hybriden Systemen das Fortschreiten der Zeit als wesentlich kontinuierlicher Vorgang im Vordergrund steht, können die Eigenschaften von SDL auch für die Spezifikation dieser Systeme genutzt werden.

Das hauptsächliche Merkmal von SDL besteht in der Spezifikationsmöglichkeit des Verhaltens eines Systems. Solch einem System wird folgende Betrachtungsweise zugrundegelegt:

- Das System ist ein offenes System.
- Das System besitzt eine Umgebung, in welche es eingebettet ist.
- Das System kommuniziert mit der Umgebung über Signale, die die Ereignisse aus der Umgebung, als auch die Reaktionen auf die Ereignisse durch das System, darstellen.

Das Verhalten des Systems wird über die Gesamtheit des Verhaltens seiner parallel arbeitenden Prozesse beschrieben. Die Prozesse, welche vollkommen gleichberechtigt nebeneinander ablaufen, stehen zueinander und zur Umgebung des Gesamtsystems durch das Senden und Empfangen von Signalen in Beziehung (Abb. 4.1). Das Empfangen eines Signals kann als ein Ereignis interpretiert werden, welches aus der Umgebung des jeweiligen Prozesses wahrgenommen wird, und das Senden eines Signals als die Reaktion des Prozesses auf ein Ereignis aus der Umgebung.

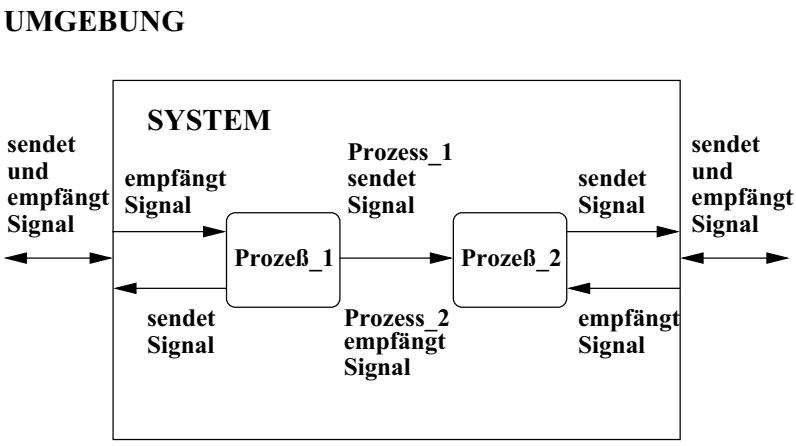


Abbildung 4.1: Verhalten eines Systems und seiner Umgebung

Die Kommunikation zwischen diesen Prozessen erfolgt in unserer Arbeit direkt und ohne Zeitverzögerung. Die Einführung einer Warteschlange für Signale, wie diese in SDL für Ereignisse geschaffen wurde, welche nicht sofort vom Empfänger verarbeitet werden können, besitzt für unseren Ansatz keine Bedeutung, da die Signale der Synchronisation von Prozessen dienen.

4.1.2 Blockstruktur und Signalwege

Die Prozesse werden auf der Grundlage des Automatenmodells, deren Übergänge durch das Senden bzw. Empfangen von Signalen unter bestimmten Bedingungen und bestimmten Zuweisungen an Variablen gekennzeichnet sind, beschrieben. Die Automaten können

als eigenständige Systeme gesehen werden, welche die kleinsten Einheiten in der Blockhierarchie darstellen. Automaten lassen sich zu Blöcken zusammenfassen, wodurch eine in ihrer Struktur abgeschlossene Einheit entsteht, welche wieder als ein vollständiges System betrachtet werden kann.
Systeme können ineinander geschachtelt werden. Diese Eigenschaft bietet folgende Vorteile:

- Systeme können hierarchisch über Subsysteme spezifiziert werden.

- Eine derartige Hierarchie erhöht die Lesbarkeit der Spezifikation.

- Das System wird durch diese Hierarchie in verschiedene Abstraktionsstufen eingeteilt. Die Prozesse bilden dabei die unterste Stufe, welche alle Einzelheiten für die Kommunikation zwischen den Prozessen enthält. Das System selbst stellt die oberste Stufe, in welcher nur die Kommunikationspfade zwischen den Subsystemen als Blackboxen erkennbar sind, dar.

- Subsysteme können als separate Typen ausgelagert werden. Diese Typen lassen sich in weitere Systeme einbinden und durch Instantiierungen mehrfach nutzen.

- Typen können durch Parametrisierung als abstrakte Typen genutzt werden.

Die Prozesse und Blöcke sind über Signalwege verbunden, welche auch über die Grenzen dieser Einheiten Signale weiterleiten können. In SDL werden zwischen Signalwegen über Blockgrenzen hinaus und zwischen Signalwegen, welche Prozesse innerhalb eines Blockes verbinden, bezüglich ihrer Zeitverzögerung unterschieden. Im Unterschied zu SDL geht unser Ansatz davon aus, dass bei den Vorgängen des Sendens und Empfangens von Signalen keine Zeitverzögerungen auftreten, da die Signale, wie bereits angesprochen, im wesentlichen der Synchronisation des Prozessverhaltens dienen.
In Abbildung 4.2 ist die mögliche Struktur eines Systems, welches in SDL beschrieben werden kann, aufgeführt. Auf jeder Hierarchiestufe können sich einzelne Prozesse, als auch Blöcke befinden, die weiterhin Blöcke und Prozesse enthalten können. Die in doppelten Umrandungen dargestellten Blöcke und Prozesse bilden Instanzen von Typstrukturen, welche auf jeder beliebig übergeordneten Ebene definiert sein können oder sich außerhalb der Spezifikation in einer extra Bibliothek befinden, wie dies bei 'Block_5' der Fall ist. Die Zahlen in der rechten unteren Ecke einer jeden Instanziierung kennzeichnen die Anzahl der Instanzen. Die durch mehrere Pfeile dargestellten Signalwege, auch Kanäle genannt, weisen auf eine Besonderheit zur Realisierung der Modularisierung hin. Die Auslagerung der Blöcke und Prozesse zur Wiederverwendung in weiteren Systemen verlangt die Definition einer klaren Schnittstelle der Signalwege an den Grenzbereichen dieser Einheiten.

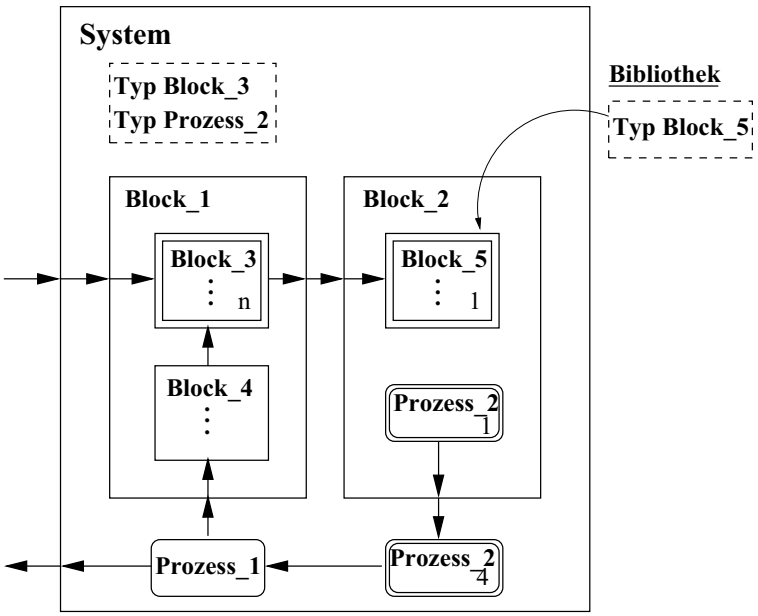


Abbildung 4.2: Schachtelung, Modularisierung und Signalwege

4.2 Kommunizierende hierarchische Zustandsautomaten

Neben SDL wurde unser Ansatz zur Entwicklung der Sprachen MODEL-HS und VYSMO durch kommunizierende hierarchische Zustandsautomaten beeinflusst. Der Ursprung solcher Automaten ist in den Statecharts von [Har87, HPSS87] zu finden. Hier werden erstmals hierarchisch geschachtelte, endliche Zustandsautomaten vorgestellt, welche neben einfachen Zuständen sogenannte *Superzustände* enthalten, die selbst endliche Zustandsautomaten darstellen. Dieser Ansatz erlaubt knappere Repräsentationen von Zustandsautomaten, wobei sich 2 Vorteile der Beschreibung ergeben:

1. schrittweise Verfeinerung als Mechanismus zur Strukturierung,
2. gemeinsame Nutzung und Wiederverwendung derselben Zustandsautomaten in unterschiedlichem Kontext.

Statecharts fanden ihre Anwendung in objekt-orientierten Methoden und Sprachen der Softwareentwicklung wie ROOM [SGW94] und UML [BRJ98], wobei Statecharts dort erst einmal nur auf syntaktischer Ebene zum Softwareentwurf eingesetzt wurden. Im Laufe der Jahre wurden für UML formale Semantikdefinitionen auf der Basis von zum Beispiel:

- kommunizierenden, erweiterten Zeitautomaten [GOO04],
- Transitionssystemen [Var02],

- Petrinetzen [Ham05]

entwickelt. Später wurden Echtzeitaspekte [DJPV02, GH04, BCD+06, RSM06] und die Integration hybrider Systeme [Por01, BKK02, BGO04] für UML mit Blick auf formale Verifikationsverfahren [GHK00, BGHS04, GOO04, YLWD07] berücksichtigt. Besonderheiten eingebetteter Systeme sind in UML [Gra08] als auch anderen auf Statecharts basierenden Sprachen wie CHARON [ADE+03, AIK+03] integriert worden.
Mit der Entwicklung formaler Sematiken für Statecharts wurden gleichzeitig kompositionale Aspekte [GLL99, LvdBC00, vdB00, QX04] in Bezug auf Nebenläufigkeit und Kommunikation zwischen den Zustandsautomaten eingeführt. Unser Ansatz lehnt sich insbesondere an Erkenntnisse aus Vorarbeiten [AKY99, AG04] und Arbeiten [AGH+00, AGLS01, AGLS06] zur Spezifikation hybrider Systeme mit CHARON an. Im Gegensatz zu SDL ist dadurch eine zusätzliche Hierarchisierung von Lokationen in unseren hybriden Automaten möglich. Der Automat zu 'Prozess_2' aus der Abbildung 4.2 kann dadurch zum Beispiel dem Schema aus Abbildung 4.3 entsprechen. Lokationen können wie 'Lokation_1' ohne weitere hierarchische Untergliederung auftreten bzw. wie 'Lokation_2' auf Blöcken bzw. kommunizierenden Automaten sowie 'Lokation_3' auf Prozessen bzw. hierarchischen Automaten basieren. Theoretisch ist dementsprechend eine beliebige Schachtelung von kommunizierenden und hierarchischen Automaten möglich.

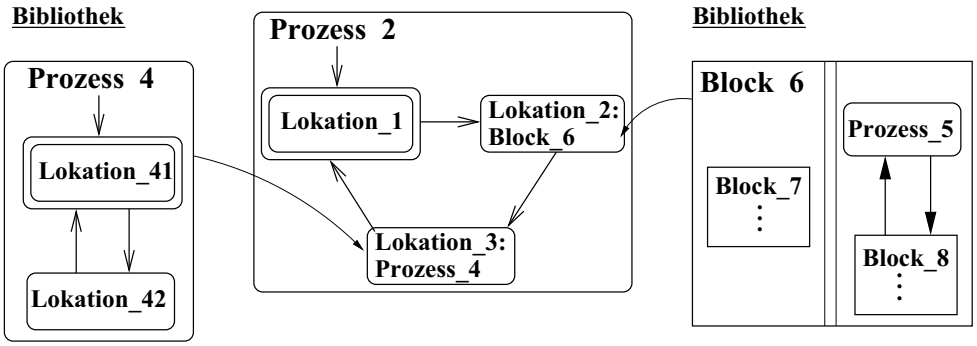


Abbildung 4.3: Kommunizierende hierarchische Automaten

Während 'Prozess_4' einen hierarchischen Automaten bildet, der auf den einfachen Lokationen 'Lokation_41' und 'Lokation_42' basiert, stellt 'Block_6' einen kommunizierenden Automat dar, in welchem 'Block_7' vollständig parallel ohne den Austausch von Signalen zu 'Prozess_5' und 'Block_8' abläuft. 'Prozess_5' und 'Block_8' dagegen kommunizieren intern über die dargestellten Signalwege. Transitionen zwischen Lokationen sind dabei mit einfachen Pfeilen gekennzeichnet und Signalwege zwischen Blöcken und Prozessen mit gefüllten Pfeilspitzen.
In der Abbildung 4.4 sind Komplexitätsaussagen zur Komprimierung und Ausdrucksstärke der Beschreibungen kommunizierender hierarchischer Automaten zu finden, die in [AKY99] auf der Grundlage von Sprachfamilien $\mathcal{L} \quad \{L_n | n = 1, 2, ...\}$ nachgewiesen und graphisch zusammengefasst wurden. Zur Akzeptanz der Sprachfamilien wird die

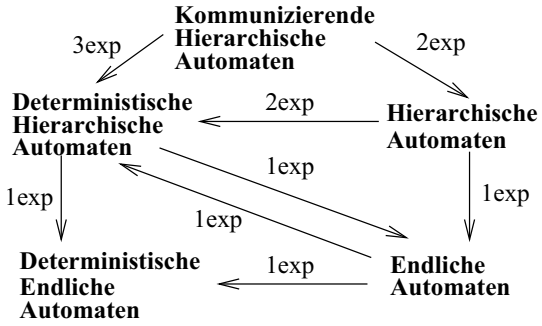


Abbildung 4.4: Komplexität verkürzter Beschreibungen

notwendige Anzahl an Zuständen in den Automaten betrachtet. Hierarchische Automaten können mit exponentiellem Aufwand in endliche Automaten überführt werden. Diese Aussage gilt auch im Fall deterministischer Automaten. Ein nichtdeterministischer endlicher Automat kann in exponentiellem Maße verkürzt zum deterministischen endlichen Automaten auftreten. Ein hierarchischer Automat kann dagegen sogar in doppelt exponentiellem Maße zum deterministischen hierarchischen Automaten auftreten. Deterministische hierarchische Automaten können exponentiell knapper als nichtdeterministische endliche Automaten sein. Da Nichtdeterminismus und Hierarchie keine vergleichbaren Erweiterungen sind, kann auch umgekehrt ein nichtdeterministischer endlicher Automat exponentiell verkürzt gegenüber dem Ergebnis einer Überführung in einen deterministischen hierarchischen Automaten auftreten. Kommunizierende hierarchische Automaten können in doppelt exponentiellem Maße verkürzt zu nichtdeterministischen hierarchischen Automaten (bzw. nichtdeterminischen endlichen Automaten) auftreten und dreifach exponentiell knapper als deterministische hierarchische Automaten (bzw. deterministische endliche Automaten) sein.
Probleme, die mit dem Aufbau modularer Strukturen über kommunizierenden hierarchischen Automaten und hybriden Automaten, die parallele und sequentielle Kompositionen zulassen, zusammenhängen und Einfluss auf Schlussfolgerungstechniken zur Abstraktion, kompositionaler Verfeinerung und Prüfung einzelner Eigenschaften unter vorherigen Annahmen (assumed-guarantee reasoning) besitzen, sind in [AKY99, AG04] und [AGH$^+$00, AGLS01, AGLS06] erarbeitet worden. In Verbindung mit den Unterschieden zu unserem Ansatz der symbolischen Simulation hybrider Systeme werden solche Probleme in Abschnitt 5.3.6 genauer betrachtet.

4.3 Hierarchisch hybride Automaten

Die folgenden Definitionen, welche hier für unseren Ansatz eingeführt werden, und die Überführung von hierarchisch und synchronisierend hybriden Automaten in flache Automaten sind zusätzlich zu nachfolgenden kleinen Beispielen im Anhang E Abschnitt E.3

anhand eines vollständigen Beispiels anschaulich illustriert.
Hierarchisch hybride Automaten zeichnen sich neben der Beschreibung des Verhaltens durch eine Menge verschiedenartiger Parameter, die in Schnittstellen bzw. lokalen Deklarationsteilen spezifiziert sein können, aus. Dabei sind Parameter nach Größen zur Beschreibung der physikalischen Umwelt und des Automatisierungssystems einerseits und nach Parameter zur Programmierung hybrider Systeme im Sinn der symbolischen Simulation in CLP andererseits zu unterscheiden. Physikalische und zum Automatisierungssystem gehörige Größen sind:

- festgelegte, nicht änderbare Parameter für Läufe (z.B Anfangstemperatur oder -druck), die nur in Schnittstellen deklariert sein können,
- Uhren zur Spezifikation lokalen und globalen zeitlichen Fortschritts, die in Schnittstellen und lokalen Deklarationsteilen deklariert sein können sowie
- kontinuierliche und diskrete Variablen zur Spezifikation physikalischer, kontinuierlich fortschreitender Größen (z.B. Temperatur, Geschwindigkeit) und sich diskret ändernder Größen (z.B. Zählvorgänge), die in Schnittstellen und lokalen Deklarationsteilen deklariert sein können.

Parameter, die der Programmierung hybrider Systeme zur symbolischen Simulation in CLP dienen, sind:

- Steuervariablen zur Einsparung von Transitionen und Lokationen, die in Schnittstellen und lokalen Deklarationsteilen deklariert sein können sowie
- Signale zur Symbolisierung von Ereignissen, die durch ihre Nutzung zur reinen Synchronisation bzw. zur Synchronisation mit zusätzlicher Verfeinerungsfunktion unterschieden werden und nur in den Schnittstellen deklariert sein können.

Die Beschreibung des Verhaltens der hierarchisch hybriden Automaten enthält Besonderheiten wie:

- Lokationen zum Warten auf eine Aktivierung beschriebener Prozesse, die keine explizit beschriebene Invariante und Aktivitäten aufweisen und standardmäßig vor dem Prozess, der durch den hierarchisch hybriden Automaten spezifiziert ist, erscheinen,
- Lokationen, die einfache Lokationen darstellen und nicht verfeinert werden können sowie komplexe Lokationen, die auf Automaten aus Bibliotheken zur weiteren Verfeinerung basieren,
- die Einteilung der Übergangsrelation in Übergangsrelationen, welche sich in der Art der Quell- bzw. Ziellokation unterscheiden und somit aus technischer Sicht die Transformation hierarchisch hybrider Automaten in flache Automaten bezüglich des Akzeptanzverhaltens in hierarchischen Strukturen unterstützen und

- die Einteilung der Endlokationen in Endlokationen ohne nachfolgende Lokation, welche selbst nicht verfeinert werden können, da diese Endlokationen bei der Verfeinerung übergeordneter Automaten reine Platzhalterfunktionen für den zeitlichen Verlauf nach der Abarbeitung der Verfeinerung besitzen und bei der Transformation in flache Automaten entfallen, sowie in Endlokationen mit nachfolgender Lokation, für die eine weitere Verfeinerung zulässig ist.

In formaler Beschreibung sind die genannten Fakten in der folgenden Definition enthalten.

Definition 4.3.1
Ein **hierarchisch hybrider Automat** *HHA* ist ein Tupel $\langle Name, Interface, Declaration, Behaviour\rangle$, wobei neben dem Namen $Name$ des Automaten *HHA* dessen Schnittstelle $Interface$, lokaler Deklarationsteil $Declaration$ und Verhalten $Behaviour$ beschrieben wird.

Definition 4.3.2
Die **Schnittstelle** *Interface* ist ein Tupel $\langle HHAInv, HHAAct, FormPara, HHASig\rangle$, wobei die Elemente des Tupels folgende Bedeutung besitzen:

- $HHAInv$ ist ein formaler Parameter für eine vom HHA einzuhaltende Invariante. Ist die Invariante nicht explizit belegt, so besitzt die Invariante den Wert 'true'.

- $HHAAct$ ist ein formaler Parameter für eine Menge von Aktivitäten, unter denen $Invariant$ einzuhalten ist.

- $FormPara$ tragen Informationen in Form von Daten, die zur Kommunikation von außen an den HHA bzw. von dem HHA nach außen an die Umgebung übergeben werden, und wird unterteilt in:

  $Para$ ist eine endliche Menge von formalen Parametern, deren Werte zur Laufzeit an einen Lauf gebunden und nicht verändert werden können.

  $SClock$ ist eine Menge von Uhren, die nicht lokal vorliegen und in mehreren Automaten den Takt angeben.

  $SVar$ ist ein Tupel, bestehend aus $\langle SContinuous, SDiscrete, SControl\rangle$, deren Elemente nicht lokal deklariert sind und folgende Bedeutung besitzen:

  * $SContinuous$ ist eine Menge von Variablenparametern, deren Werte sich kontinuierlich ändern können,
  * $SDiscrete$ ist eine Menge von Variablenparametern, deren Werte sich diskret ändern können,
  * $SControl$ ist eine Menge von Variablenparametern, deren Werte sich diskret ändern können und die der Einsparung von Transitionen und Lokationen dienen.

- $HHASig$ symbolisieren Ereignisse, die zur Synchronisation von außen an den HHA bzw. von dem HHA nach außen an die Umgebung übergeben werden, und ist unterteilt in:

- $SynchronSig$ ist die Menge von Signalen, die zu denjenigen Transitionen gehören, welche nicht zu Endlokationen führen und der Synchronisation mit anderen Automaten dienen. $SynchronSig$ unterteilt sich in In und Out, wobei jedes $s_i \in In$ ein empfangenes Signal und jedes $s_o \in Out$ ein gesendetes Signal darstellt.

- $RefineSig$ ist die Menge von Signalen, die zu denjenigen Transitionen gehören, welche zu Endlokationen führen und neben der Synchronisation der Verfeinerung von Transitionen dienen. $RefineSig$ unterteilt sich in In und Out, wobei jedes $s_i \in In$ ein empfangenes Signal und jedes $s_o \in Out$ ein gesendetes Signal darstellt.

Definition 4.3.3
Der **lokale Deklarationsteil** *Declaration* bildet ein Tupel $\langle Clock, Var \rangle$, wobei die Elemente des Tupels folgende Bedeutung besitzen:

- $Clock$ ist eine Menge von lokalen Uhren.

- Var ist eine endliche Menge von lokalen Variablen, die unterteilt ist in:

  $Continuous$ für sich kontinuierlich ändernde Variablen,

  $Discrete$ für sich diskret ändernde Variablen und

  $Control$ für sich diskret ändernde Variablen mit dem Ziel der Einsparung von Lokationen und Transitionen.

Definition 4.3.4
Das **Verhalten** *Behaviour* wird durch ein Tupel $\langle L, \delta, l_{wait}, \delta_0, l_0, F, Guard, Action, Inv, Act, HHAs, Actual \rangle$ gebildet, wobei die Elemente des Tupels folgende Bedeutung besitzen:

- L ist die Menge der Lokationen, welche in die disjunkten Mengen EL und KL unterteilt ist:

  EL ist eine endliche Menge von $einfachen\ Lokationen$.

  KL ist eine endliche Menge von $komplexen\ Lokationen$.

- δ ist die Übergangsrelation, welche in 4 Arten von Übergangsrelationen bezüglich der Mengen von Lokationen EL und KL unterteilt ist:

  δ_{ee} ist eine $Übergangsrelation$ mit $\delta \subseteq EL \times Guard \times Action \times EL$.

  δ_{ek} ist eine $Übergangsrelation$ mit $\delta \subseteq EL \times Guard \times Action \times KL$.

  δ_{ke} ist eine $Übergangsrelation$ mit $\delta \subseteq KL \times Guard \times Action \times EL$.

δ_{kk} ist eine $\ddot{U}bergangsrelation$ mit $\delta \subseteq KL \times Guard \times Action \times KL$.

Für jede Lokation $l \in EL \cup KL$ gibt es in Form $(l, \langle \emptyset, true \rangle, \langle \emptyset, Id \rangle, l)$ einen $faulen$ $\ddot{U}bergang$, wobei Id eine Funktion ist, die jeder Uhr $c \in Clock$ und jeder Variablen $v \in Var$ die Identitätsfunktion zuordnet.

- $l_{wait} \in EL$ ist eine ausgezeichnete Hilfslokation, in der ein Automat auf seine Aktivierung warten muss. Die Aktivierung erfolgt, wenn erforderliche Signale $r \in Refine.In \cup Synch.In$ der Anfangsübergangsrelation δ_0 anliegen und die lineare Formel ψ des Wächters der Anfangsübergangsrelation δ_0 erfüllt ist. Während der Aktivierung können Signale $s \in Refine.Out \cup Synch.Out$ ausgesandt und Werte von Uhren und Variablen neu gesetzt werden. Liegt kein Signal r an und wurde keine Bedingung ψ explizit für δ_0 spezifiziert, so ist ψ durch den Wahrheitswert $true$ auswertbar und der Übergang zu l_0 erfolgt ohne Verzögerung mit dem Start der Systemzeit. In l_{wait} genügt die Zeit t immer der Invariante: $0 \leq t$. Die Zeit schreitet entsprechend des Taktes der Systemzeit fort.

- $\delta_0 \quad l_{wait} \times Guard \times Action \times l_0$ ist die $Anfangsübergangsrelation$.

- $l_0 \in EL \cup KL$ ist die $Anfangslokation$.

- $F \subseteq L$ ist die Menge der Endlokationen, welche in die disjunkten Mengen TF und FF unterteilt ist:

  $TF \subseteq EL$ ist die Menge von Endlokationen, wobei für alle $l \in TF$ gilt, es gibt keinen Übergang $l \times Guard \times Action \times EL \cup KL$, d.h. l besitzt keine nachfolgende Lokation.

  $FF \subseteq EL \cup KL$ ist die Menge von Endlokationen, wobei für alle $l \in FF$ gilt, es gibt mindestens einen Übergang $l \times Guard \times Action \times EL \cup KL$, d.h. l besitzt mindestens eine nachfolgende Lokation.

- $Guard$ ist eine Menge von Wächtern, wobei $g \in Guard$ ein Tupel $\langle GR, \psi \rangle$ mit $GR \subseteq RefineSig.In \cup SynchSig.In$ und ψ eine lineare Formel ist.

- $Action$ ist eine Menge von Aktionen, wobei $a \in Action$ ein Tupel $\langle AS, f \rangle$ mit $AS \subseteq RefineSig.Out \cup SynchSig.Out$ und f ist eine Funktion, die jeder Uhr $c \in Clock$ und jeder Variablen $v \in Var$ eine lineare Funktion zuordnet.

- Inv ist eine Funktion, die jeder Lokation $l \in L$ eine $Invariante$ zuweist, welche eine lineare Formel ϕ ist.

- Act ist eine Funktion, die jeder Lokation $l \in L$ eine Menge von $Aktivitäten$ zuweist, welche in konjunktiver Beziehung zueinander stehen. Eine $Aktivität$ wird in Form einer Funktion $x \# f(t)$ angegeben, mit $x \in \{SClock, SVar, Clock, Var\}$, t ist eine Zeitvariable und $f(t)$ ist ein Term in Abhängigkeit von der Zeit und $\# \in \{<, \leq, =, \geq, >\}$.

- $HHAs$ ist eine Funktion, die jeder komplexen Lokation $l \in KL$ einen HHA zuordnet.

- $Actual$ ist eine Funktion, die jeder komplexen Lokation $l \in KL$ eine Menge von aktuellen Parameterwerten zuordnet.

4.3.1 Hierarchisierung

Die Grundlage für hierarchisch hybride Automaten bildet der hier genutzte Begriff der Hierarchisierung.

Begriff 4.3.1
Als **Hierarchisierung** wird der Vorgang bezeichnet, der in hierarchisch hybriden Automaten die Bildung von Hierarchieebenen aufgrund der Verfeinerung komplexer Lokationen durch bereits in Bibliotheken vorliegende, hierarchisch hybride Automaten im Sinn einer sequentiellen Komposition bewirkt.

An dieser Stelle soll auf die Frage eingegangen werden, in welcher Art und Weise hierarchisch hybride Automaten hierarchisiert werden können bzw. welche Voraussetzungen ein hierarchisch hybrider Automat erfüllen muss, um eine Verfeinerung für eine komplexe Lokation eines bestehenden hierarchisch hybriden Automaten zu bilden. Hierbei sind der strukturelle Aufbau des gegebenen Automaten und die Semantik im Sinne der Ausführbarkeit von symbolischen Läufen zur Hierarchisierung zu unterscheiden. In [AGLS06] werden syntaktische Voraussetzungen der Verfeinerung von Modi, als welche hierarchisch hybride Automaten in der Sprache CHARON beschrieben werden, mit dem Begriff der Kompatibilität zwischen den Modi bezeichnet. Die Kompatibilität wird dabei durch gemeinsame Variablen sowie gemeinsame Eingangs- und Ausgangspunkte der Modi festgelegt. Die Verfeinerung selbst wird durch die Menge der möglichen Läufe angegeben.
Unser Ansatz geht davon aus, dass Automaten, die der Hierarchisierung dienen, bereits in Bibliotheken als Modelle für vollständig beschriebene Prozesse vorhanden sind und für die Akzeptanz von Wörtern notwendige Endlokationen besitzt.
Aufgrund der Modellierung mit Hilfe hybrider Automaten, bei welcher die Prozesse über unendlich lange Zeiträume betrachtet werden, enthalten Automaten, die der Verfeinerung einer komplexen Lokation dienen, nicht nur Lokationen und Transitionen, die die komplexe Lokation selbst verfeinern, sondern darüber hinaus Endlokationen und Transitionen zu den Endlokationen, die eine Aussage über den weiteren Verlauf nach dem Verlassen der komplexen Lokation treffen. Wie in Abbildung 4.5 bildet hierbei die Menge aller Transitionen des verfeinernden Automaten, welche zu Endlokationen führen, eine Verfeinerung der Menge all derjenigen Transitionen, welche in der zu verfeinernden komplexen Lokation ausgelöst werden und von dieser wegführen. Durch die gestrichelten Linien ist gekennzeichnet, dass die Transition mit der Bedingung 'Wächter_1' durch die beiden Transitionen mit den Bedingungen 'Wächter_b' und 'Wächter_c' verfeinert wird. Die Endlokationen des verfeinernden Automaten zeigen, unter welchen Bedingungen, spezifiziert

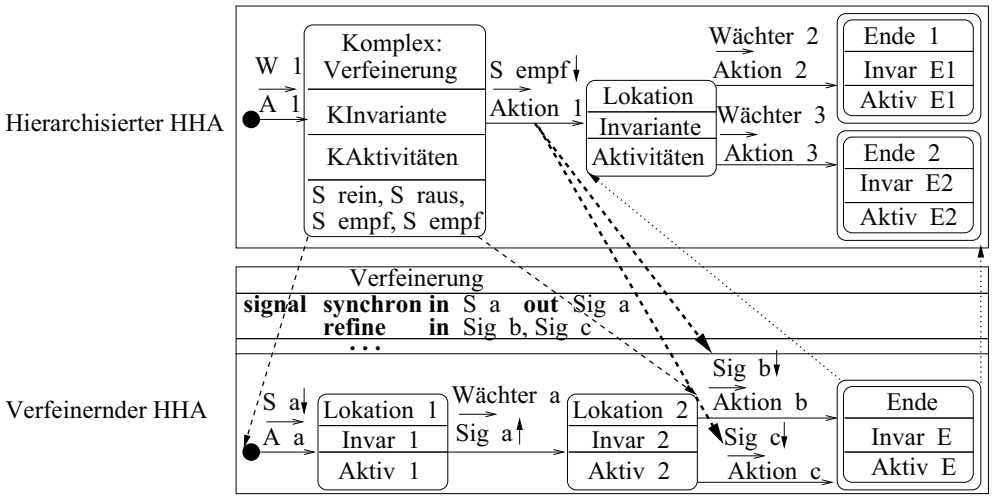


Abbildung 4.5: Verfeinerung einer komplexen Lokation

als Invarianten der Endlokationen, und in welchem kontinuierlichen Verlauf, spezifiziert als Aktivitäten der Endlokationen, sich die Größen der Umgebung nach der Beendigung des verfeinernden Prozesses entwickeln. Als Voraussetzung gilt, dass sich die Bedingungen und der kontinuierliche Verlauf aus dem vorher ausgeführten verfeinernden Prozess ergeben und keiner weiteren Beschränkung unterliegen, da die Endlokationen als reine Platzhalter angesehen werden, deren Invarianten und Aktivitäten nicht noch einmal für nachfolgende Aktionen und Verläufe der verfeinerten Lokationen geprüft werden. Endlokationen sollen damit auf Garantie ohne nochmalige Überprüfung eine Abstraktion für alle möglichen Fortsetzungen, die der verfeinerten komplexen Lokation folgen können, bilden. Mit den gepunkteten Linien ist in Abbildung 4.5 gekennzeichnet, dass die Endlokation 'Ende' des verfeinernden Automaten somit den zeitlichen Verlauf von 'Lokation' bis zu 'Ende_1' bzw. 'Ende_2' des hierarchisierten hybriden Automaten symbolisiert.
Auf diese Art und Weise wird das von uns genutzte Prinzip der Akzeptanz von Symbolen an Transitionen unterstützt, welches der Untersuchung von Echtzeitsystemen auf der Grundlage formaler Sprachen dient. Nicht Lokationen oder Zustände, wie in [AGLS06] oder bei den Statecharts [HPSS87, LvdBC00, SASX05], bilden die Schnittstellen der Automaten, welche mit hin- und wegführenden Übergängen verbunden werden, sondern Transitionen, die durch Signale gekennzeichnet sind, welche an den Schnittstellen der hybriden Automaten zur Synchronisation und Hierarchisierung ausgetauscht werden. Dabei verfolgen wir die Methode von SDL [EHS97, MT01], wobei Kanäle zur Beschreibung der Schnittstellen und Verfeinerung genutzt werden. In Anlehnung an diese Kanäle werden unsere Transitionen beschrieben.
Im Unterschied zu bestehenden Modellen von Automaten [dAHS02, AG04, AGLS06, HM06], in denen Verfeinerungsprinzipien untersucht werden, wird in unserem Ansatz das Akzeptanzverhalten näher betrachtet, da das Ergebnis unserer symbolischen Simulation von der Akzeptanz des als Folge von Signalen verbunden mit symbolischen Größen

der Umgebung beschriebenen Verhaltens abhängt. Zur Kennzeichnung des Akzeptanzverhaltens existieren in unseren Modellen 2 Arten von Endlokationen:

1. Endlokationen, denen keine weiteren Lokationen folgen und die nicht verfeinert werden können, d.h. zu den einfachen Lokationen gehören und

2. Endlokationen, denen weitere Lokationen folgen und die verfeinert werden können.

Zur Schaffung eines wohldefinierten Akzeptanzverhaltens für unsere hierarchisch hybriden Automaten, welches bisher noch in keinem weiteren Ansatz aufgeführt ist, wird von folgender Begriffsbildung ausgegangen.

Begriff 4.3.2
Ein hierarchisch hybrider Automat besitzt ein **wohldefiniertes Akzeptanzverhalten**, wenn für die Verfeinerung aller komplexen Lokationen, welche:

1. selbst Endlokationen darstellen, gilt:

   - Signalfolgen, die bis zu der Endlokation akzeptiert wurden, müssen auch nach der Verfeinerung der Endlokation, wenn auch innerhalb eingeschränkter Zeiträume, akzeptierbar sein. Die Anfangslokation der Verfeinerung muss dementsprechend eine Endlokation darstellen können.
   - Folgen von Signalen, die in dem verfeinernden hierarchisch hybriden Automaten A bis zu einer Endlokation, welcher weitere Lokationen folgen, akzeptiert wurden, sollen bei der Verfeinerung als Suffix einer Folge F von Signalen dann akzeptierbar sein, wenn die verfeinerte komplexe Lokation kL eine Endlokation ist. Die Folge F von Signalen bildet die Signalfolge, welche von der Anfangslokation bis zur verfeinerten komplexen Lokation kL verläuft.

2. selbst keine Endlokationen darstellen, gilt:

   - Akzeptanzverhalten, welches durch Endlokationen im verfeinernden hierarchisch hybriden Automaten der komplexen Lokation kL festgelegt wurde, wird auf der Ebene von kL nicht zugelassen. Dieses Vorgehen entspricht dem Prinzip der Verfeinerung. Die Verfeinerung kann das Akzeptanzverhalten einschränken, jedoch nicht erweitern. Bestehen auf der zu verfeinernden Ebene komplexe Lokationen über Mengen von akzeptierenden Zuständen innerhalb eines Zeitraumes, so können die Mengen der akzeptierenden Zustände in der Verfeinerung auf ein oder mehrere kleinere in diesem Zeitraum enthaltene Zeitintervalle eingeschränkt werden. Wird jedoch auf der zu verfeinernden Ebene kein Zustand als akzeptierender Zustand ausgewiesen, so ist das Akzeptanzverhalten der Verfeinerung nur für die Verfeinerung selbst, nicht aber für die zu verfeinernde Ebene relevant und entfällt bei der Betrachtung des gesamten Prozesses, der sich aus der sequentiellen Komposition ergibt.

Aus Sicht der Syntax und der statischen Semantik müssen die komplexe Lokation *kL* des zu hierarchisierenden HHA und der zugeordnete hierarchisch hybride Automat *A* zur Verfeinerung von *kL* im Sinn *wohldefinierter Schnittstellen* für unseren Ansatz folgende Merkmale aufweisen:

Syntax:

- Für jeden formalen Parameter des Automaten *A* ist in der komplexen Lokation *kL* genau ein aktueller Parameter erklärt.
- Jedem Signal im Abschnitt 'RefineSig' der Schnittstelle des Automaten *A* ist genau ein Signal der Wächter bzw. Aktionen der wegführenden Transitionen der komplexen Lokation *kL* zugeordnet.
- Umgekehrt muss jedem Signal der Wächter bzw. Aktionen der wegführenden Transitionen der komplexen Lokation *kL* wenigstens ein Signal des Abschnittes 'RefineSig' der Schnittstelle des Automaten *A* zugeordnet sein.

Statische Semantik:

- Die Zuordnung der Signale ist in der Art vorzunehmen, dass zwischen den Signalen an den hinführenden Transitionen zu Endlokationen des Automaten *A*, die im Abschnitt 'RefineSig' deklariert sind, und der Menge von Signalen jeder wegführenden Transition der komplexen Lokation *kL* eine homomorphe Abbildung *h* entsteht:
  SigA Signalmenge einer hinführenden Transition zu einer Endlokation des Automaten *A*,
  SigAll Menge von Signalmengen *SigA*,
  SigkL Signalmenge einer wegführenden Transition der komplexen Lokation *kL*,
  $h : SigAll \rightarrow SigkL$,
  $\forall\, SigAll_i \in SigAll \wedge SigAll_j \in SigAll$: $SigAll_i \cap SigAll_j \quad \emptyset$.
- Die Bedingungen der Wächter und Zuweisungen der Aktionen aller hinführenden Transitionen zu Endzuständen des verfeinernden Automaten *A*, die derselben wegführenden Transition der komplexen Lokation *kL* zugeordnet sind, müssen mit den Bedingungen der Wächter und den Zuweisungen der Aktionen der wegführenden Transition konsistent sein.

Semantisch bildet ein hierarchisch hybrider Automat *A* eine Verfeinerung der komplexen Lokation *kL*:

1. wenn jeder symbolische Lauf des Automaten *A*, welcher eine Folge von Signalmengen verbunden mit symbolischen Werten für die Größen der physikalischen Umgebung darstellt, die Invariante der komplexen Lokation *kL* unter Berücksichtigung der Aktivitäten in *kL* erfüllt und

2. die Konjunktion der Bedingungen der Wächter aller hinführenden Transitionen zu Endzuständen des verfeinernden Automaten *A*, die derselben wegführenden Transition der komplexen Lokation *kL* zugeordnet sind, gerade die Bedingung des Wächters der wegführenden Transition der komplexen Lokation *kL* bilden sowie die Zuweisungen der Aktionen aller hinführenden Transitionen zu Endzuständen des verfeinernden Automaten *A*, die derselben wegführenden Transition der komplexen Lokation *kL* zugeordnet sind, mit den Zuweisungen der Aktion der wegführenden Transition der komplexen Lokation *kL* übereinstimmen.

4.3.2 Semantik eines HHA

Die Semantik eines HHA lässt sich auf Grundlage der Struktur eines flachen Automaten $fHHA$ mit folgender Tupeldarstellung beschreiben:
$fHHA = \langle fL, fS, fR, fl_0, fTF, fFF, fParam, fClock, fVar, fGuard, fAction, f\delta, fAct, fInv \rangle$:

- fL ist die Menge der Lokationen.

- fS ist die Menge der gesendeten Signale.

- fR ist die Menge der empfangenen Signale.

- fl_0 ist die Anfangslokation.

- fTF ist die Menge der Endlokationen ohne Nachfolgelokation.

- fFF ist die Menge der Endlokationen mit Nachfolgelokation.

- $fParam$ ist die Menge der nicht veränderbaren fest vorgegebenen Werte.

- $fClock$ ist die Menge von Uhren, welche ausgezeichnete kontinuierliche Variablen bilden.

- $fVar$ ist die Menge der Variablen.

- $fGuard$ ist die Menge der Wächter an Transitionen.

- $fAction$ ist die Menge der Aktionen an Transitionen.

- $f\delta = fL \times fGuard \times fAction \times fL$ ist die Übergangsrelation.

- $fInv$ ist die Funktion, welche jeder Lokation $l \in fL$ eine Invariante zuordnet.

- $fAct$ ist die Funktion, welche jeder Lokation $l \in fL$ eine Menge von Aktivitäten zuordnet.

Diese Struktur wird entsprechend eines rekursiven Algorithmus aufgebaut. Der Hauptfunktion 'flat_HHA' werden:

1. die Beschreibung des hierarchisch hybriden Automaten in HHA,
2. eine Invariante $SysInv$, welche der HHA einzuhalten hat,
3. eine Menge $SysAct$, die Aktivitäten für kontinuierliche Größen der Umgebung enthält, die für den gesamten HHA gelten und unter denen die Invariante $SysInv$ eingehalten werden muss sowie
4. aktuelle Parameter $ActualPara$, die die konkreten Werte für die formal spezifizierten Deklarationen in der Schnittstelle des HHA darstellen,

als Parameter übergeben. Mit der Funktion 'flat_HHA' wird ein flacher Automat aus dem hierarchisch hybriden Automaten HHA abgeleitet. Aus technischen Gründen wurde zur Verminderung der Anzahl an Zuordnungsfunktionen eine Umordnung der Parameter des HHA wie folgt vorgenommen:

$HHA \quad \langle Name, Interface, Behaviour \rangle$
$Interface \quad \langle HHAInv, HHAAct, Para, SClock, SVar, SynchronSig, RefineSig \rangle$
$Behaviour \quad \langle Simple, Complex, General \rangle$
$Simple \quad \langle EL, \delta_{ee} \rangle$
$Complex \quad \langle KL, \delta_{ek}, \delta_{ke}, \delta_{kk}, HHAs, Actual \rangle$
$General \quad \langle l_{wait}, \delta_0, l_0, TF, FF, Clock, Var, Guard, Action, Inv, Act \rangle$.

Dabei bildet $Simple$ den Anteil, welcher sich aus Lokationen und Transitionen ergibt, die nur mit einfachen Lokationen ohne Verfeinerungsmöglichkeit zusammenhängen. Im

Tupel $Complex$ sind die komplexen Lokationen selbst, mit diesen Lokationen zusammenhängende Transitionen, die Zuordnungsfunktion von verfeinernden HHA zu komplexen Lokationen und zugehörige aktuelle Parameter aufgeführt. Allgemeine Informationen, die den gesamten Automaten bzw. einfache als auch komplexe Lokationen betreffen, sind in $General$ zusammengefasst. Somit wird der rekursiv ausgeführte Algorithmus in Pseudocode wie folgt angegeben:

define function flat_HHA ($HHA, SysInv, SysAct, ActualPara$)
let HHA $\langle Name, Interface, \langle Simple, Complex, \langle l_{wait}, \delta_0, l_0,$
$TF, FF, Clock, Var, Guard, Action, Inv, Act\rangle\rangle\rangle$ **in**
clean(transform($HHA, SysInv, SysAct, ActualPara$),$FF$);

In der Funktion 'flat_HHA' liefert der Aufruf von 'transform' einen flachen Automaten. Die unten aufgeführte Funktion 'transform' ist eine rekursiv ausgeführte Funktion. Der einfachste Fall liegt vor, wenn keine komplexen Lokationen in KL existieren. In diesem Fall erfolgt eine Zuordnung der Bestandteile des vorliegenden Automaten zu den Bestandteilen eines flachen Automaten. Sind komplexe Lokationen vorhanden, so wird die erste komplexe Lokation aus KL in l zwischengespeichert. Zu der komplexen Lokation l wird der flache Automat mit 'flat_HHA' ermittelt und mit dem flachen Automaten des verbleibenden hierarchischen Automaten durch 'concat' verknüpft. Der verbleibende hierarchische Automat besteht dabei aus allen Lokationen bis auf l und allen Transitionen bis auf die Transitionen, die mit l verbunden und in $\langle \delta_{lek}, \delta_{lke}, \delta_{lkk} \rangle$ gespeichert sind. Zur korrekten Verknüpfung der flachen Automaten der einzelnen Lokationen l mit den flachen Automaten, die in 'transform' aus den verbleibenden Restautomaten berechnet wurden, mussten Verknüpfungsmerkmale wie i) Markierungen von Lokationen als Endlokationen mit Nachfolgelokationen, ii) Endlokationen ohne Nachfolgelokationen und iii) Transitionen, die zu solchen Endlokationen ohne Nachfolgelokationen führen, erhalten bleiben. Aus technischen Gründen wurden diese Merkmale nicht separat gespeichert, sondern in den flachen Automaten beibehalten. Zum Schluss der gesamten Transformation sind diese nur der Transformation dienenden Kennzeichen mit der Funktion 'clean' zu löschen.

define function transform($\langle Name, Interface, \langle Simple,$
$\langle KL, \delta_{ek}, \delta_{ke}, \delta_{kk}, HHAs, Actual\rangle, General\rangle,$
$SysInv, SysAct, ActualPara$)
if KL {} **then**
simple_case($Name, Interface, Simple, General, SysInv, SysAct,$
$ActualPara, \langle \emptyset, \emptyset, \emptyset,$noname,$\emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset,$
Id,Tautology$\rangle$);
else
let l first(KL) **and** R rest(KL) **and**
$\langle \delta_{lek}, \delta_{lke}, \delta_{lkk} \rangle$ transitions($\delta_{ek}, \delta_{ke}, \delta_{kk}, l$) **and**
$General$ $\langle l_{wait}, \delta_0, l_0, TF, FF, Clock, Var, Guard, Action, Inv, Act\rangle$ **and**
$HHAs(l)$ $\langle lName, lInterface, \langle lSimple, lComplex, \langle ll_{wait}, l\delta_0, ll_0, TF,$

$lFF, lClock, lVar, lGuard, lAction, lInv, lAct\rangle\rangle$ **in**
concat(clean(transform($HHAs(l), Inv(l), Act(l), Actual(l)), lFF$),
transform($\langle Name, Interface, \langle Simple,$
$\langle R, \delta_{ek}, \delta_{ke}, \delta_{kk}, HHAs, Actual\rangle, General\rangle\rangle$,
$SysInv, SysAct, ActualPara$),
$l, \delta_{lek}, \delta_{lke}, \delta_{lkk}, Interface, General, SysInv, SysAct, ActualPara$);

Der gesamte Algorithmus ist im Anhang A zu finden. Dort sind die Funktionen 'clean', 'simple_case' und 'concat' vollständig implementiert. In der Funktion 'flat_HHA' transformiert der Algorithmus einen gegebenen HHA mit dessen Invariante $SysInv$, zugewiesener Menge von Aktivitäten $SysAct$ und aktuellen Parametern $ActualPara$ in einen flachen Automaten. Der flache Automat muss bezüglich des im Abschnitt 4.3.1 beschriebenen wohldefinierten Akzeptanzverhaltens und einer wohldefinierten Schnittstelle verschiedene Eigenschaften besitzen. Eine Eigenschaft ergibt sich aus dem geforderten Akzeptanzverhalten, wobei gilt:
Jede *absolute Anfangslokation* eines verfeinernden hierarchisch hybriden Automaten einer komplexen Lokation, die im betrachteten HHA eine Endlokation mit Nachfolgelokationen ist, bildet selbst eine Endlokation mit Nachfolgelokationen im flachen Automaten des betrachteten HHA. Die absolute Anfangslokation eines hierarchisch hybriden Automaten $VHHA$ zur Verfeinerung lässt sich mit der Funktion 'abs_anfang' wie folgt berechnen:

define function abs_anfang($VHHA$)
if $VHHA.\delta_0 \neq \langle l_{wait}, \langle\{\}, C\rangle, \langle\{\}, A\rangle, l_0\rangle$ **then**
$VHHA.l_{wait}$
else if $VHHA.l_0 \in VHHA.EL$ **then**
$VHHA.l_0$
else abs_anfang($HHAs(VHHA.l_0)$);

Wenn die Standardlokation l_{wait} zum Warten auf eine Aktivierung von $VHHA$ nicht sofort verlassen wird, da Signale an der Anfangstransition δ_0 anliegen müssen, so wird l_{wait} die absolute Anfangslokation. Sonst wird überprüft, ob die definierte Anfangslokation l_0 aus $VHHA$ eine einfache Lokation ist. Wenn dies der Fall ist, wird l_0 die absolute Anfangslokation, anderenfalls ist die absolute Anfangslokation des die Lokation l_0 zu verfeinernden hierarchisch hybriden Automaten zu berechnen.
Mit der Funktion 'all_abs_anfang' wird nun die Menge aller absoluten Anfangslokationen, die den Verfeinerungen aller komplexen Endlokationen mit Nachfolgelokationen eines betrachteten HHA entstammen, berechnet:

define function all_abs_anfang($VHHAs$)
if $VHHAs = \{\}$ **then**
$\{\}$
else let $VHHAf$ first($VHHAs$) **and** $VHHArest$ rest($VHHAs$) **in**
list(abs_anfang($VHHAf$), all_abs_anfang($VHHArest$));

Die Menge $VHHAs$ stellt sämtliche, den komplexen Endlokationen des betrachteten HHA zugeordnete, verfeinernde, hierarchisch hybride Automaten dar. Ist diese Menge leer, so ist die Menge der absoluten Anfangslokationen leer. Sonst wird für den ersten hierarchisch hybriden Automaten $VHHAf$ die absolute Anfangslokation berechnet und durch 'list' zur Restliste der berechneten Anfangslokationen der verbliebenen Automaten aus $VHHArest$ hinzugefügt. Im Folgenden kann somit folgende Behauptung aufgestellt werden.

Lemma 4.3.1
Die Funktion 'flat_HHA' berechnet für jeden beliebigen hierarchisch hybriden Automaten $HHA \quad \langle Name, Interface, Behaviour\rangle$ in Verbindung mit einer aktuellen Invariante $SysInv$, aktuellen Aktivitäten $SysAct$ und aktuellen Parametern $ActualPara$ einen flachen Automaten $fHHA$ der Struktur $\langle fL, fS, fR, fl_0, fTF, fFF, fParam, fClock, fVar, fGuard, fAction\rangle$, für welchen gilt:
Wenn $VHHAs$ die Menge sämtlicher, den komplexen Endlokationen von HHA zugeordneten, verfeinernden, hierarchisch hybriden Automaten darstellt, so ist:
all_abs_anfang($VHHAs$) $\subseteq fFF$.

Der Beweis des Satzes wird durch das Prinzip der allgemeinen Induktion geführt. Dabei wird vorausgesetzt, dass mit der Funktion 'clean' alle in einem flachen Automaten während der Transformation entstandenen Endlokationen, die nur eine technische Bedeutung für die Transformation, nicht aber für das Endergebnis besitzen, gelöscht werden. Die Funktion 'simple_case' ermittelt zu jedem Automaten ohne komplexe Lokationen den flachen Automaten und die Funktion 'concat' bildet aus dem flachen Automaten einer komplexen Lokation L und dem flachen Automaten des restlichen HHA ohne die komplexe Lokation L einen einheitlich flachen Automaten. Als trivial wird angenommen, dass all_abs_anfang($VHHAs$) eine echte Teilmenge von fFF sein kann, da zusätzlich zu absoluten Anfangslokationen weitere Endlokationen mit Nachfolgelokationen aus den verfeinernden Automaten erhalten bleiben können. In der Induktionsvoraussetzung wird für eine komplexe Lokation l mit dem zugeordneten hierarchisch hybriden Automaten $aHHAs(l)$ angenommen, dass dessen absolute Anfangslokation der Anfangslokation lfl_0 des durch 'transform' gebildeten flachen Automaten entspricht. Diese Voraussetzung wurde hier nicht explizit bewiesen. Der Beweis kann auf der Basis der Funktion 'abs_anfang' erbracht werden, welche in der Funktion 'transform' direkt implementiert ist.

Beweis 4.3.1
Seien für die Menge $HHAall$ der hierarchisch hybriden Automaten folgende Elemente gegeben:

Basiselemente: *be* $\langle Name, Interface, \langle Simple, \langle \emptyset, \emptyset, \emptyset, \emptyset, HHA, Actual\rangle, General\rangle$
sind hierarchisch hybride Automaten, welche keine komplexen Lokationen besitzen.

Erzeugte Elemente: Wenn $a \in HHAall$ und $b \in HHAall$, wobei
$a \quad \langle Name, Interface, \langle Simple, \langle KL, \delta_{ek}, \delta_{ke}, \delta_{kk}, HHAs, Actual\rangle, General\rangle$,
dann $ee \in HHAall$ mit $ee \quad \langle Name, Interface, \langle Simple, \langle KL \cup l, \delta_{ek} \cup \delta_{lek}, \delta_{ke} \cup \delta_{lke}, \delta_{kk} \cup \delta_{lkk}, HHAs, Actual\rangle, General\rangle$ und $HHAs(l) \quad b$
wobei δ_{lek}, δ_{lke} und δ_{lkk} die Transitionen sind, die die Lokation l mit bereits definierten einfachen und komplexen Lokationen verbinden.

Behauptung: Für einen beliebigen $c \in HHAall$ gilt:
flat_HHA($c, SysInv, SysAct, Actual$) $\langle fL, fS, fR, fl_0, fTF, fFF, fParam, fClock, fVar, fGuard, fAction, f\delta, fAct, fInv\rangle$,
wobei
$c \quad \langle Name, Interface, \langle Simple, \langle KL, \delta_{ek}, \delta_{ke}, \delta_{kk}, HHAs, Actual\rangle, \langle l_{wait}, \delta_0, l_0, TF, FF, Clock, Var, Guard, Action, Inv, Act\rangle\rangle\rangle$
und $\forall\ (klff_i \in KL \wedge klff_i \in FF)$ mit $i = 1, ..., n$ gilt:
all_abs_anfang($\{HHAs(klff_1), ..., HHAs(klff_n)\}$) $\subseteq fFF$

Beweis:
flat_HHA($c, SysInv, SysAct, Actual$)
clean(transform($c, SysInv, SysAct, ActualPara$),$FF$).
clean($\langle fL, fS, fR, fl_0, afTF, afFF, fParam, fClock, fVar, fGuard, fAction, af\delta, fAct, fInv\rangle, FF$)
$\langle fL, fS, fR, fl_0, fTF, fFF, fParam, fClock, fVar, fGuard, fAction, f\delta, fAct, fInv\rangle$
erzeugt aus einem flachen Automaten A einen flachen Automaten B durch:
Entfernung folgender Endlokationen und Transitionen aus $afTF$, $afFF$ und $af\delta$:
1. Endlokationen ohne Nachfolgelokationen aus Verfeinerungen,
2. Endlokationen mit Nachfolgelokationen, die aus Verfeinerungen komplexer Lokationen entstanden sind, welche selbst keine Endlokationen darstellen und
3. Transitionen, die zu Endlokationen ohne Nachfolgelokationen führen.

Es ist noch zu zeigen, dass der Aufruf von transform($c, SysInv, SysAct, ActualPara$) zu einem flachen Automaten der Form $\langle fL, fS, fR, fl_0, fTF, fFF, fParam, fClock, fVar, fGuard, fAction, f\delta, fAct, fInv\rangle$ mit :
$\forall\ (klff_i \in KL \wedge klff_i \in FF)$ mit $i = 1, ..., n$ und
all_abs_anfang($\{HHAs(klff_1), ..., HHAs(klff_n)\}$) $\subseteq fFF$ führt.

Teilbeweis:

Induktionsanfang:
Für alle be als Basiselemente gilt:
transform($be, SysInv, SysAct, ActualPara$)
simple_case($Interface, Simple, General, SysInv, SysAct, ActualPara,$
$StartAutom$)
mit
$be \quad \langle Name, Interface, \langle Simple, \langle \emptyset, \emptyset, \emptyset, \emptyset, HHA, Actual \rangle, General \rangle$, wobei
$StartAutom$ einen flachen Automaten ohne Inhalt als Ausgangspunkt zur Berechnung des zu be zugehörigen flachen Automaten darstellt
simple_case($Interface, Simple, General, SysInv, SysAct, ActualPara,$
$StartAutom$)
$\langle fL, fS, fR, fl_0, fTF, fFF, fParam, fClock, fVar, fGuard, fAction, f\delta,$
$fAct, fInv \rangle$
mit all_abs_anfang($\{\}$) $\quad \{\} \subseteq fFF$

Induktionsvoraussetzung:
Für einen beliebigen HHA:
$a \quad \langle aName, aInterface, \langle aSimple, \langle aKL, a\delta_{ek}, a\delta_{ke}, a\delta_{kk}, aHHAs, aActual \rangle,$
$\langle al_{wait}, a\delta_0, al_0, aTF, aFF, aClock, aVar, aGuard, aAction, aInv, aAct \rangle \rangle \rangle$ und
einer beliebigen Invariante $SysInv$, beliebigen Aktivitäten $SysAct$ und beliebigen aktuellen Parametern $ActualPara$ gelte:
transform($a, SysInv, SysAct, ActualPara$)
$\langle afL, afS, afR, afl_0, afTF, afFF, afParam, afClock, afVar, afGuard,$
$afAction, af\delta, afAct, afInv \rangle$
mit $\forall$ ($klff_i \in aKL \wedge klff_i \in aFF$) mit $i = 1, ..., n$ gelte:
all_abs_anfang($\{aHHAs(aklff_1), ..., aHHAs(aklff_n)\}$) $\subseteq afFF$ und
Für eine komplexe Lokation l, die noch nicht zum HHA a gehört, gelte:
transform($aHHAs(l), SysInv, SysAct, ActualPara$)
$\langle lfL, lfS, lfR, lfl_0, lfTF, lfFF, lfParam, lfClock, lfVar, lfGuard,$
$lfAction, lf\delta, lfAct, lfInv \rangle$
mit: abs_anfang($aHHAs(l)$) $\quad lfl_0$

Induktionsbehauptung:
Dann gilt für einen HHA:
$c \quad \langle Name, Interface, \langle Simple, \langle aKL \cup l, a\delta_{ek} \cup \delta_{lek}, a\delta_{ke} \cup \delta_{lke}, a\delta_{kk} \cup \delta_{lkk},$
$aHHAs, Actual \rangle, \langle l_{wait}, \delta_0, l_0, TF, FF, Clock, Var, Guard, Action, Inv, Act \rangle \rangle \rangle$ und
eine beliebige Invariante $SysInv$, beliebige Aktivitäten $SysAct$ und beliebige aktuelle Parameter $ActualPara$:
transform($c, SysInv, SysAct, ActualPara$)
$\langle fL, fS, fR, fl_0, fTF, fFF, fParam, fClock, fVar, fGuard, fAction, f\delta,$
$fAct, fInv \rangle$

mit $\forall\ (klff_i \in aKL \wedge klff_i \in FF)$ mit $i\ =\ 1, ..., n$ und $l \notin FF$ gelte:
all_abs_anfang($\{aHHAs(aklff_1), ..., aHHAs(aklff_n)\}$) $\subseteq fFF$ bzw.
wenn $l \in FF$, dann:
all_abs_anfang($\{aHHAs(l), aHHAs(aklff_1), ..., aHHAs(aklff_n)\}$) $\subseteq fFF$

Induktionsbeweis:

Wenn $General$ $\langle l_{wait}, \delta_0, l_0, TF, FF, Clock, Var, Guard, Action, Inv, Act\rangle$ und
$ConcatPara$ $(l, \delta_{lek}, \delta_{lke}, \delta_{lkk}, Interface, General, SysInv, SysAct, ActualPara)$,
dann ist:

nach Funktionsdefinition
transform($c, SysInv, SysAct, ActualPara$)
transform($\langle Name, Interface, \langle Simple, \langle aKL \cup l, a\delta_{ek} \cup \delta_{lek}, a\delta_{ke} \cup \delta_{lke}, a\delta_{kk} \cup \delta_{lkk},$
$aHHAs, Actual\rangle, General\rangle\rangle, SysInv, SysAct, ActualPara$)
concat(clean(transform($aHHAs(l), Inv(l), Act(l), Actual(l)$),$FF$),
transform($\langle Name, Interface, \langle Simple, \langle KL, (\delta_{ek}, \delta_{ke}, \delta_{kk}, aHHAs,$
$Actual\rangle, General\rangle\rangle, SysInv, SysAct, ActualPara$),
$ConcatPara$)
concat(clean(transform($aHHAs(l), Inv(l), Act(l), Actual(l)), FF$),
transform($a, SysInv, SysAct, ActualPara$),
$ConcatPara$)

nach Induktionsvoraussetzung
concat(clean($\langle lfL, lfS, lfR, lfl_0, lfTF, lfFF, lfParam, lfClock, lfVar,$
$lfGuard, lfAction, lf\delta, lfAct, lfInv\rangle, FF$),
$\langle afL, afS, afR, afl_0, afTF, afFF, afParam, afClock, afVar,$
$afGuard, afAction, af\delta, afAct, afInv\rangle$,
$ConcatPara$)

nach Funktionsdef. von 'clean'
Wenn EoN, EmN und $EoN\delta$ Endlokationen und Transitionen sind, welche durch 'clean' gelöscht wurden, so gilt weiterhin:
concat($\langle lfL, lfS, lfR, lfl_0, lfTF/EoN, lfFF/EmN, lfParam, lfClock,$
$lfVar, lfGuard, lfAction, lf\delta/EoN\delta, lfAct, lfInv\rangle$,
$\langle afL, afS, afR, afl_0, afTF, afFF, afParam, afClock, afVar,$
$afGuard, afAction, af\delta, afAct, afInv\rangle$,
$ConcatPara$)

nach Funktionsdef. von 'concat'
flacher Automat:
$\langle fL, fS, fR, fl_0, fTF, fFF, fParam, fClock, fVar, fGuard, fAction,$
$f\delta, fAct, fInv\rangle$

mit
if $l \in FF$ **then**
fFF vereinige($afFF$, $\{lfl_0\}$)
else fFF afFF
endif;

nach Induktionsvoraussetzung
abs_anfang($aHHAs(l)$) lfl_0, so gilt:
if $l \in FF$ **then**
vereinige(fFF, $\{abs\_anfang(aHHAs(l)\}$)
else fFF afFF
endif;
und aus $\forall\ (klff_i \in aKL \wedge klff_i \in aFF)$ mit $i = 1, ..., n$:
all_abs_anfang($\{aHHAs(aklff_1), ..., aHHAs(aklff_n)\}$) $\subseteq afFF$
folgt:
if $l \in FF$ **then**
all_abs_anfang($\{aHHAs(l), aHHAs(aklff_1), ..., aHHAs(aklff_n)\}$) $\subseteq afFF$
else all_abs_anfang($\{aHHAs(aklff_1), ..., aHHAs(aklff_n)\}$) $\subseteq afFF$;
endif;

□

Semantisch können Inkonsistenzen in den Invarianten sowie bei den Bedingungen an den Transitionen und bezüglich der Aktivitäten in den Lokationen sowie Aktionen an den Transitionen auftreten. Diese Inkonsistenzen und Widersprüche können erst durch die symbolische Simulation nachgewiesen werden. Denkbar ist der Einsatz der symbolischen Simulation im Verlauf der Transformation eines HHA in einen flachen Automaten, um mögliche Modelländerungen während der Transformation zuzulassen.
Eine iterativ-rekursive Variante des Algorithmus befindet sich im Anhang B. Hier werden während des iterativen Aufbaus des flachen Automaten Zwischenergebnisse in Matrizen und Variablen zur Wiederverwendung gespeichert. Durch die Speicherung von Zwischenergebnissen brauchen am Ende keine Transitionen und Endzustände im aufgebauten Automaten, die nur der Zwischeninformation dienen, gelöscht werden.

4.3.3 Beispiel einer Hierarchisierung

Anhand einer Prüfung, die nur einen Prüfungsversuch zulässt, ist in den Abbildungen 4.6, 4.7 und 4.8 ein Beispiel gegeben, welches die genannten Merkmale der Verfeinerung noch einmal verdeutlicht. Die Abbildungen sind in der Sprache VYSMO beschrieben, welche in Kapitel 6 vorgestellt wird. Die 'Einfache Pruefung' enthält eine Schnittstelle, die aus den Signalen, welche der reinen Synchronisation dienen, und den Signalen, welche zusätzlich zur Verfeinerung bestimmt sind, besteht. Eine lokale Uhr 'X' symbolisiert die Zeit, die während des Prozesses der einfachen Prüfung fortschreitet und zurückgesetzt

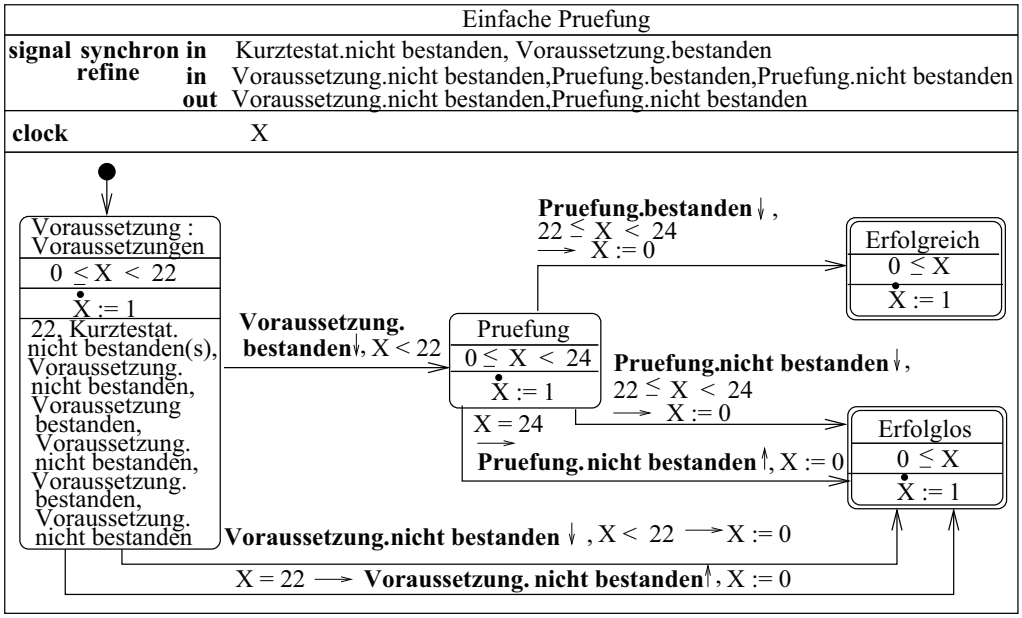


Abbildung 4.6: Prüfungsablauf mit zu verfeinernder Voraussetzung

werden kann. In Abbildung 4.6 ist im unteren Abschnitt der Verlauf der Prüfung mit Hilfe eines hierarchisch hybriden Automaten beschrieben. Zum Ablegen der Prüfung muss in einem Zeitraum von 22 Monaten eine Voraussetzung erfüllt sein, welche hier noch nicht näher spezifiziert ist. Erst durch die Instantiierung eines weiteren hybriden Automaten 'Voraussetzungen', siehe Abbildung 4.7, wird gezeigt, wie die Voraussetzung für einen konkreten Fall aussehen kann. Dadurch erreichen wir eine Verfeinerung der Lokation 'Voraussetzung', welche zusammen mit dem Automaten in Abbildung 4.6 den flachen hybriden Automaten der Abbildung 4.8 ergibt. Nachdem die Voraussetzung erfolgreich bestanden wurde, darf die Prüfung abgelegt werden. Die Prüfung *muss* innerhalb von 24 Monaten angegangen werden. Das ist die Bestimmung der Prüfungsordnung. Laut Durchführungsordnung gibt es jedoch einen festgelegten Prüfungszeitraum, in der ein Student die Prüfung ablegen *kann*. Dieser Zeitraum von 22 bis 24 Monaten ist mit den wegführenden Transitionen des Prüfungsversuches 'Pruefung' zu den folgenden Zeiträumen 'Erfolgreich' und 'Erfolglos' als Bedingung verbunden. Werden die Zeiträume, welche in den Lokationen 'Voraussetzung' und 'Pruefung' als Invarianten ausgeführt sind, nicht eingehalten, so gelten die Voraussetzung bzw. die Prüfung automatisch durch das eintretende Zeitereignis von 'X 22' bzw. 'X 24' als nicht bestanden. Die Signale 'Voraussetzung.nicht bestanden' und 'Pruefung.nicht bestanden' werden dann als Informationen an die Umgebung gesendet, welches durch die nach oben gerichteten Pfeile ausgedrückt wird. Alle anderen Signale werden von der Umgebung empfangen, welches durch die nach unten gerichteten Pfeile gekennzeichnet ist.
Die Voraussetzung ist nach einem Prozess zu absolvieren, der mit Hilfe des hierarchisch

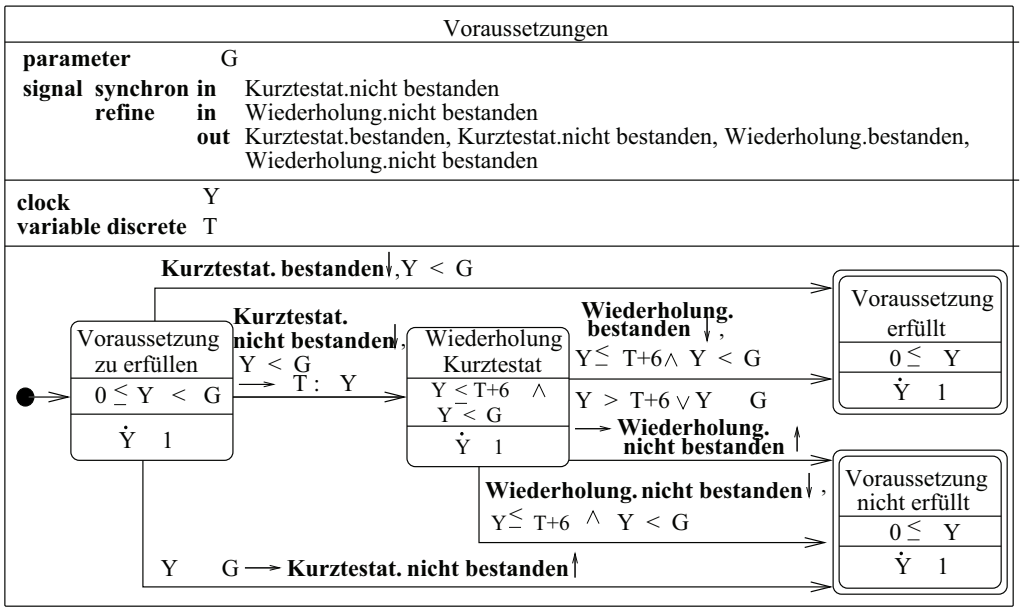


Abbildung 4.7: Verfeinerte Voraussetzung

hybriden Automaten in Abbildung 4.7 beschrieben ist. Dieser Prozess ist mit einem Parameter 'G' ausgestattet, der entsprechend der Umgebung, in welchem der Prozess abläuft, die zeitliche Grenze für das Erfüllen einer Voraussetzung angibt. Wie in Lokation 'Voraussetzung' der Abbildung 4.6 zu sehen ist, wird der Parameter 'G' hier bei der Instantiierung mit dem Wert '22' belegt. Der Automat besitzt eine Uhr 'Y' zur Messung der lokalen Zeit und eine diskrete Variable 'T', die den Zeitpunkt enthält, zu welchem die Voraussetzung zum ersten Mal nicht bestanden wurde. Als Voraussetzung ist ein Kurztestat innerhalb von 22 Monaten zu bestehen. Wurde das Kurztestat beim ersten Versuch mit Erfolg bestanden, so gilt die Voraussetzung für jeden beliebig folgenden Zeitpunkt als erfüllt. Wurde der erste Versuch nicht bestanden, so kann ein Wiederholungsversuch in den nächsten 6 Monaten, spätestens jedoch bis zu der zeitlichen Grenze von 22 Monaten erfolgen. Die Invariante von 'Wiederholung Kurztestat' und die Bedingungen an den wegführenden Transitionen der Lokation 'Wiederholung Kurztestat' beschreiben diese Aussagen mit den Teilausdrücken 'Y $\leq$ T+6' bzw. 'Y > T+6' und 'Y < G' bzw. 'Y G'. Wurde der Wiederholungsversuch bestanden, so ist die Voraussetzung für immer erfüllt, sonst ist die Voraussetzung nie mehr erfüllbar und der Automat verbleibt für jeden beliebig folgenden Zeitpunkt in der Lokation 'Voraussetzung nicht erfüllt'.
Der in Abbildung 4.8 aufgeführte Automat entsteht durch das Aufheben der Hierarchiestufe in der Lokation 'Voraussetzung'. Hierbei werden die Merkmale einer Hierarchisierung für hybride Automaten ausgenutzt. Jedem formalen Parameter des Automaten 'Voraussetzungen' war in der Lokation 'Voraussetzung' des Automaten 'Einfache Pruefung' genau ein aktueller Parameter zugeordnet worden. Dabei tritt in der Zuordnung der Signale das Signal 'Kurztestat.nicht bestanden' auf, welches in 'Einfache Pruefung' nicht

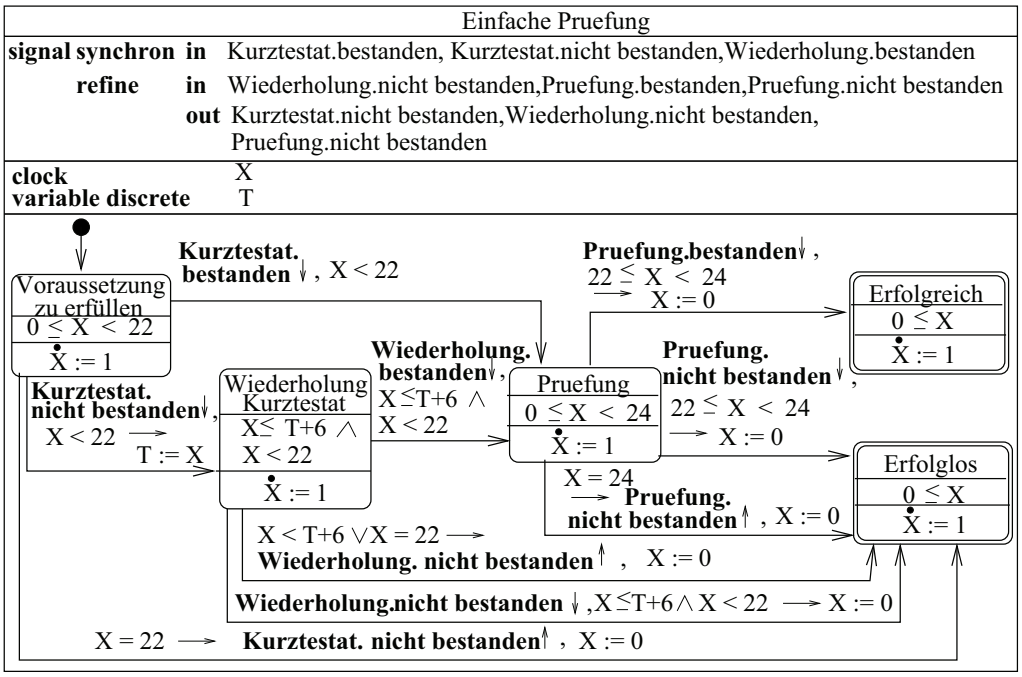


Abbildung 4.8: Aus sequentieller Komposition entstandener flacher Automat

explizit definiert ist. Das Signal wurde mit einem ’(s)’ gekennzeichnet, um zu zeigen, dass ’Kurztestat.nicht bestanden’ nicht mit der Verfeinerung im Zusammenhang steht, sondern der Synchronisation mit der Umgebung dient. In Verfeinerungen anderer Automaten wie zum Beispiel Statecharts ist solch ein Signal nach außen nicht sichtbar. Doch in unserem Ansatz verfolgen wir neben der Hierarchisierung das Prinzip der Synchronisation mittels gesendeter und empfangener Signale. Das heißt, neben der Hierarchisierung können hybride Automaten über die parallele Komposition verbunden sein, bei der auch intern betrachtete Signale der zur Hierarchisierung eingesetzten Automaten für die Synchronisation von Bedeutung sein können. Solche Signale müssen durch Kopiervorgänge in die Schnittstellen der Automaten mit der zu verfeinernden Lokation als für die Umgebung sichtbare Parameter aufgenommen werden. Die Endlokationen ’Voraussetzung erfüllt’ und ’Vorausetzung nicht erfüllt’ des zur Verfeinerung eingesetzten Automaten der Voraussetzungen entfallen bei der Aufhebung der Hierarchie. Das Verhalten, welches in diesen Endlokationen beschrieben wurde, wird durch das Verhalten der Lokationen ’Pruefung’, ’Erfolgreich’ und ’Erfolglos’ sowie der Transitionen zwischen den Lokationen überlagert. Dieses Merkmal ergibt sich aus der Modellierung vollständig hybrider Automaten, welche zur Verfeinerung komplexer Lokationen in einem erweiterten Kontext eingesetzt werden. Durch den Einsatz vollständig hybrider Automaten zur Verfeinerung ist nicht nur die komplex zu verfeinernde Lokation selbst von der Verfeinerung betroffen, sondern auch alle nachfolgenden Transitionen der komplexen Lokation. In unserem Beispiel werden die Transitionen mit den Signalen ’Voraussetzung.bestanden’ und ’Voraussetzung.nicht

bestanden' durch Transitionen mit den Signalen 'Kurztestat.bestanden', 'Kurztestat.nicht bestanden', 'Wiederholung.bestanden' und 'Wiederholung.nicht bestanden' in Kombination mit empfangendem und sendendem Charakter verfeinert. Diese Verfeinerung entspricht der Verfeinerung von Kanälen in SDL. Die Invariante '$0 \leq X < 22$' der komplexen Lokation 'Voraussetzung' wird durch die Lokationen und Transitionen des Automaten 'Voraussetzungen' nicht verletzt. Auch die Bedingungen an den Transitionen mit den Signalen 'Kurztestat.bestanden', 'Kurztestat.nicht bestanden', 'Wiederholung.bestanden' und 'Wiederholung.nicht bestanden' sind zu den Bedingungen der Transitionen mit den Signalen 'Voraussetzung.bestanden' und 'Voraussetzung.nicht bestanden' konsistent und die Aktionen des Zurücksetzens der Uhren entsprechen einander, wodurch hier auf den Erhalt zweier Uhren verzichtet wurde. Im Allgemeinen bleiben zur Übersetzung hierarchisch hybrider Automaten in flache Automaten lokal definierte Uhren der Verfeinerungen unter Umbenennung erhalten.

4.4 Synchronisierend hybride Automaten

Nachdem hierarchisch hybride Automaten zur sequentiellen Komposition ausführlich dargestellt wurden, wird im Folgenden der Aufbau und die Arbeitsweise des synchronisierend hybriden Automaten zur parallelen Komposition erklärt. Auch diese Automaten besitzen die Beschreibung einer Schnittstelle, die neben dem Namen, einem Deklarationsteil für lokal genutzte Bezeichner und einem Synchronisationsteil den Austausch von festen, nicht änderbaren Parametern, Uhren- und Variablenwerten sowie Signalen ähnlich der Beschreibung hierarchisch hybrider Automaten des Abschnittes 4.3 zulässt. Dabei werden Signale zur Synchronisation importiert und exportiert.
Der lokale Deklarationsteil im SHA weist mit der Beschreibung empfangener und gesendeter Signale eine Besonderheit auf, wodurch Signale deklariert sind, welche nicht in die Umgebung des SHA gelangen. Unter einer gegebenen Bezeichnung empfängt der SHA solche Signale von internen Automaten und sendet diese ausschließlich, auch unter Umbenennung, an weitere, intern spezifizierte Automaten.
Im Synchronisationsteil existiert neben den eigentlichen Synchronisationsverbindungen in $Connect$, welche die Verbindungen zwischen den einzelnen Unterautomaten des SHA darstellen, eine ausgezeichnete Synchronisationsverbindung $Connect_0$, welche eine Verbindung von der Umgebung zum SHA darstellt, die für alle Unterautomaten zu Beginn der Laufzeit des Systems gültig ist. Alle Synchronisationsverbindungen können mit Wächtern und Aktionen, die Bedingungen und Zuweisungen beinhalten, ausgestattet sein. Diese Wächter und Aktionen wirken bei der Durchführung von Transformationen in flache Automaten und bei der Ausführung der symbolischen Simulation in Verbindung mit den Wächtern und Aktionen, die an den zugehörigen Transitionen der verbundenen Unterautomaten auftreten.

Definition 4.4.1
Ein **synchronisierend hybrider Automat** *SHA* ist ein Tupel $\langle Name, Interface, Decla-$

$ration, Synchronisation\rangle$, wobei neben dem Namen $Name$ des Automaten *SHA* dessen Schnittstelle $Interface$, lokaler Deklarationsteil $Declaration$ und Synchronisationsteil $Synchronisation$ beschrieben wird.

Definition 4.4.2
Die **Schnittstelle** *Interface* ist ein Tupel $\langle SHAInv, SHAAct, FormPara, SHASig\rangle$, wobei die Elemente des Tupels folgende Bedeutung besitzen:

- $SHAInv$ ist ein formaler Parameter für eine vom SHA einzuhaltende Invariante. Ist die Invariante nicht explizit belegt, so besitzt die Invariante den Wert 'true'.
- $SHAAct$ ist ein formaler Parameter für eine Menge von Aktivitäten, unter denen $SHAInv$ einzuhalten ist.
- $FormPara$ tragen Informationen in Form von Daten, die zur Kommunikation von außen an den SHA bzw. von dem SHA nach außen an die Umgebung übergeben werden, und wird unterteilt in:

  $Para$ ist eine endliche Menge von formalen Parametern, deren Werte nicht verändert werden können.

  $SClock$ ist eine Menge von Uhren, die nicht lokal vorliegen und in mehreren Automaten den Takt angeben.

  $SVar$ ist ein Tupel, bestehend aus $\langle SContinuous, SDiscrete, SControl\rangle$, deren Elemente nicht lokal deklariert sind und folgende Bedeutung besitzen:

  * $SContinuous$ ist eine Menge von Variablenparametern, deren Werte sich kontinuierlich ändern können,
  * $SDiscrete$ ist eine Menge von Variablenparametern, deren Werte sich diskret ändern können,
  * $SControl$ ist eine Menge von Variablenparametern, welche unterteilt ist in $In\_automaton$ und In_block. Die Werte beider Parametertypen ändern sich diskret. Parameter des Bereiches $In\_automaton$ dienen der Einsparung von Transitionen und Lokationen in hierarchisch hybriden Automaten. Parameter des Bereiches In_block dagegen dienen der Einsparung textueller und graphischer Strukturen zur Beschreibung in synchronisierend hybriden Automaten.
- $SHASig$ ist die Menge von Signalen, die zu Synchronisationsverbindungen der Umgebung mit dem Automaten gehören, und unterteilt sich in In und Out, wobei jedes $s_i \in In$ ein empfangenes Signal aus der Umgebung und jedes $s_o \in Out$ ein gesendetes Signal an die Umgebung darstellt.

Definition 4.4.3
Der **lokale Deklarationsteil** *Declaration* bildet ein Tupel $\langle Clock, Var, Signals\rangle$, wobei die Elemente des Tupels folgende Bedeutung besitzen:

- $Clock$ ist eine Menge von Uhren.

- Var ist eine endliche Menge von $Variablen$ aus den Bereichen $\langle Continuous, Discrete, Control\rangle$, deren Beschreibung mit der Beschreibung unter $SVar$ übereinstimmt. Die Werte liegen jedoch lokal vor.

- $Signals$ ist die endliche Menge von Signalen, die lokal im SHA zwischen dessen Unterautomaten ausgetauscht werden, und unterteilt sich in:

  R ist eine endliche Menge empfangener Signale, die der SHA ausschließlich von seinen Unterautomaten empfängt.

  S ist eine endliche Menge gesendeter Signale, die der SHA ausschließlich an seine Unterautomaten sendet.

Definition 4.4.4
Der **Synchronisationsteil** *Synchronisation* ist ein Tupel $\langle HHAs, HHAEnv, Connect_0, Connect, SyGuard, SyAction, Inv, Act, Actual\rangle$, wobei die Elemente folgende Bedeutung besitzen:

- $HHAs$ ist eine endliche Menge von hierarchisch hybriden Automaten.

- $HHAEnv$ ist ein ausgezeichneter hierarchisch hybrider Automat, der als Hilfsautomat die Umgebung beschreibt. Dabei wird nur der Name angegeben. Die Tupel der Schnittstelle, lokalen Deklaration und der Synchronisation sind leer.

- $Connect_0 \subseteq HHAEnv \times SyGuard \times SyAction \times SHA$ ist eine ausgezeichnete Synchronisationsverbindung zwischen dem Automaten für die Umgebung und dem beschriebenen synchronisierend hybriden Automaten, welche die Anfangsbedingungen zum Start für alle hybriden Automaten aus $HHAs$ vorgibt.

- $Connect \subseteq HHAs \times SyGuard \times SyAction \times HHAs$ ist die Menge aller Synchronisationsverbindungen zwischen den hierarchisch hybriden Automaten, wobei alle Signale g $\in GR$ aus $\langle GR, \psi\rangle$ $SyGuard$ vom ersten HHA des Tupels empfangen werden und als a $\in AS$ aus $\langle AS, f\rangle$ $SyAction$ an den zweiten HHA des Tupels gesendet werden.

- $SyGuard$ ist eine Menge von Wächtern, wobei g $\in SyGuard$ ein Tupel $\langle GR, \psi\rangle$ mit $GR \subseteq R$ und ψ eine lineare Formel ist.

- $SyAction$ ist eine Menge von Aktionen, wobei a $\in SyAction$ ein Tupel $\langle AS, f\rangle$ mit $AS \subseteq S$ und f ist eine Funktion, die jeder Uhr $c \in Clock$ und jeder Variablen $v \in Var$ eine lineare Funktion zuweist.

- Act ist eine Funktion, die jedem synchronisierend hierarchisch hybriden Automaten $hha \in HHAs$ eine Menge von $Aktivitäten$ zuweist, welche in konjunktiver Beziehung zueinander stehen. Eine $Aktivität$ wird in Form einer Funktion $x \# f(t)$

angegeben, mit $x \in \{SClock, SVar, Clock, Var\}$, t ist eine Zeitvariable und $f(t)$ ist ein Term in Abhängigkeit von der Zeit und $\# \in \{<, \leq, =, \geq, >\}$.

- Inv ist eine Funktion, die jedem synchronisierend hierarchisch hybriden Automaten $hha \in HHAs$ eine $Invariante$ zuweist, welche eine lineare Formel ϕ ist.

- $Actual$ ist eine Funktion, die jedem hierarchisch hybriden Automaten $shha \in HHAs$ eine Menge von aktuellen Parameterwerten zuweist.

4.4.1 Synchronisation

Mit dem Begriff der hier genutzten Synchronisation ist die Grundlage für synchronisierend hybride Automaten geschaffen worden.

Begriff 4.4.1
Als **Synchronisation** wird der Vorgang bezeichnet, der die Bildung eines synchronisierend hybriden Automaten aufgrund der Abstraktion von einer Menge sich synchronisierender, bereits in Bibliotheken vorliegender, hierarchisch hybrider Automaten im Sinn der parallelen Komposition bewirkt.

Der flache Automat eines SHA kann selbst wieder den Prinzipien von Instantiierung und Umbenennung, als auch Umgebungsbedingungen durch gegebene Invarianten und Aktivitäten genügen. Der vollständige, flache Automat des SHA kann folgendermaßen erstellt werden:

1. Das *Verhalten* ergibt sich aus dem berechneten Synchronisationsprodukt der Beweisidee 4.4.1.

2. Die *Schnittstelle* zur Umgebung ergibt sich aus:

   (a) der Invariante des SHA als Invariante des flachen Automaten,

   (b) den Aktivitäten des SHA als Aktivitäten des flachen Automaten,

   (c) den Parametern, Uhren und Variablen der Schnittstelle des SHA als Parameter, Uhren und Variablen der Schnittstelle des flachen Automaten,

   (d) alle empfangenen Signale, die mit Transitionen des flachen Automaten verbunden sind, welche *nicht* zu Endlokationen führen, als zur Synchronisation empfangene Signale,

   (e) alle gesendeten Signale, die mit Transitionen des flachen Automaten verbunden sind, welche *nicht* zu Endlokationen führen, als zur Synchronisation gesendete Signale,

   (f) alle empfangenen Signale, die mit Transitionen des flachen Automaten verbunden sind, welche zu Endlokationen führen, als zur Verfeinerung empfangene Signale,

(g) alle gesendeten Signale, die mit Transitionen des flachen Automaten verbunden sind, welche zu Endlokationen führen, als zur Verfeinerung empfangene Signale.

3. Der *lokale Deklarationsteil* ergibt sich aus:

   (a) den lokalen Uhren des SHA zusammen mit den Uhren der flachen Unterautomaten als lokale Uhren des flachen Automaten,

   (b) den lokalen Variablen des SHA zusammen mit den Variablen der flachen Unterautomaten als lokale Uhren des flachen Automaten.

4. Der *Name* des flachen Automaten entspricht dem Namen des SHA.

Obgleich die Darstellung des flachen Automaten eines SHA der Repräsentation eines hierarchisch hybriden Automaten entspricht, besteht ein wesentlicher Unterschied im Verhalten der beiden Arten von Automaten. Im Synchronisationsprodukt-Automaten sind nicht alle möglichen Kombinationen von Transitionen, die massiv parallel ausgeführt werden können, explizit aufgeführt. Diese Kombinationen sind jedoch zwischen allen aufeinanderfolgenden Transitionen voneinander disjunkter Automaten zulässig und werden bei der Berechnung mit der symbolischen Simulation beachtet. Entsprechend der Vorgehensweise zur Schaffung eines flachen Automaten eines SHA ist im Abschnitt 4.4.3 ein Beispiel entstanden. Im nächsten Abschnitt wird auf den Aufbau eines flachen Automaten und dessen Bedeutung als Synchronisationsprodukt eingegangen.

4.4.2 Semantik eines SHA

Die Semantik eines synchronisierend hybriden Automaten SHA entspricht dem eines flachen Automaten $fSHA$ mit:
$fSHA = \langle fL, fS, fR, fl_0, fTF, fFF, fParam, fClock, fVar, fGuard, fAction, f\delta, fAct, fInv \rangle$,
dessen Bestandteile wie im Teil der Beschreibung des HHA erklärt sind. Zur Spezifikation der Semantik wird der Begriff der transitiven Hülle bezüglich:

1. einer ausgewählten Transition eines beliebig ausgewählten Automaten HHA aus dem gegebenen SHA, die durch:

   - die Signalmenge GR und die Bedingung $Cond$ im Wächter und
   - die Signalmenge AS und die Zuweisung $Assign$ in der Aktion

   gekennzeichnet ist und zu dem flachen Automaten $fHHA$ des hierarchisch hybriden Automaten HHA gehört, sowie

2. der Relation der Synchronisationsverbindungen $Connect$ des den HHA beinhaltenden SHA eingeführt.

Im Folgenden ist dieser Begriff als rekursive Funktion in Pseudocode spezifiziert, welche einer Fixpunktberechnung entspricht:

define function transitive_closure($HHAs_i, HHAsDiff, GR, Cond, AS, Assign,$
$Connect$)
  let $HHAs_{i+1} = HHAs_i \cup HHAsDiff$ **in**
    if $HHAs_{i+1} = HHAs_i$ **then**
      ($HHAs_i, GR, Cond, AS, Assign$)
    else
      let $HHAsDiff = [DiffFirst|DiffRest]$ **in**
        transitive_closure($HHAs_{i+1}$,
                for_all_diff($DiffRest$,
                        new_diff(synchron($Connect, DiffFirst,$
                                $GR[DiffFirst],$
                                $AS[DiffFirst], HHAs_{i+1}$),
                            $DiffFirst, \emptyset, GR, Cond, AS, Assign$),
                        $Connect, HHAs_{i+1}$),
                $Connect$);

Der Aufruf der Funktion 'transitive_closure' wird mit:

transitive_closure($\emptyset, \{HHA\}, GR[HHA], Cond, AS[HHA], Assign, Connect$)

vorgenommen. Die Menge der Automaten, welche die transitive Hülle des Automaten HHA bilden, ist zum Anfang leer. Die Differenzmenge zu der iterativ zu berechnenden Nachfolgemenge besteht zu Beginn aus dem Automaten HHA. Weiterhin ist mit $GR[HHA]$ die empfangene Signalmenge des Wächters der untersuchten Transition des HHA angegeben. Die Signalmenge $GR[HHA]$ ist eine über HHA indizierte Menge, die zu einem Feld von indizierten Mengen gehört. Aus technischer Sicht ist somit der Zugriff auf die einzelnen Mengen für die spätere Wiederverwendung gewährleistet. Die Bedingung der untersuchten Transition wird in $Cond$ übergeben. In der Signalmenge $AS[HHA]$ sind die gesendeten Signale der Aktion der untersuchten Transition enthalten. Die indizierte Bezeichnung wird wie im Fall von $GR[HHA]$ verwendet. Die Zuweisungen der Aktion werden in $Assign$ übergeben. Der aktuelle Parameter $Connect$ beinhaltet alle Synchronisationsverbindungen, die in dem SHA definiert sind, in der der betrachtete HHA einen Unterautomaten darstellt.
Ist der Fixpunkt der Iteration erreicht, so sind alle Automaten, welche zur transitiven Hülle des betrachteten HHA gehören, in $HHAs_i$ enthalten. Der Rückgabewert besteht a) in GR aus der Konkatenation aller empfangenen Signale, b) in $Cond$ aus der konjunktiven Bedingung aller Bedingungen, c) in AS aus der Konkatenation aller gesendeten Signale und d) in $Assign$ aus der Vereinigung aller Zuweisungen der Synchronisationswege und Transitionen, die zum Aufbau der transitiven Hülle führen. Ist der Fixpunkt der Berechnung noch nicht erreicht, so existieren noch Automaten des SHA, die bezüglich

der vorgegebenen Synchronisation von HHA über die Synchronisationsverbindungen mit weiteren Automaten in Verbindung stehen. All diese Synchronisationsverbindungen werden untersucht, wobei eine neue Menge zu erreichender Automaten berechnet wird. Mit dieser Menge wird die Funktion 'transitive_closure' erneut aufgerufen. Die berechnete Automatenmenge bildet die Differenzmenge zwischen $HHAs_i$ und $HHAs_{i+1}$. Ist die Differenzmenge leer, so wurden keine weiteren Automaten gefunden, mit denen sich der HHA transitiv bezüglich der untersuchten Transition synchronisiert und der Fixpunkt der Berechnung ist erreicht. Der gesamte Algorithmus, der im Anhang C zu finden ist, basiert auf der Annahme, dass alle an der Synchronisation beteiligten Automaten mit genau einer Transition zur Synchronisationsverbindung beitragen. Weitere Fälle und deren Probleme bei der Behandlung sind im Ausblick erläutert.
In SHIFT [DGS98] wurde ein ähnlicher Algorithmus entwickelt. Dieser Algorithmus geht von der Synchronisation hybrider Automaten über interne und externe Ereignisse der Transitionen aus. Die internen und externen Ereignisse können unseren gesendeten und empfangenen Signalen entsprechen. Die Synchronisation muss in SHIFT nicht notwendigerweise ausgeführt werden. In [DGS98] werden noch keine hierarchischen Strukturen berücksichtigt bzw. Umbenennungs- und Verdeckungsprinzipien beachtet.
Sei die Menge der $HHAs \quad \{HHA_1, ..., HHA_n\}$ mit den zugehörigen flachen Automaten $\{fHHA_1, ..., fHHA_n\}$, so gilt für $fSHA$:

- $fL \subseteq fHHA_1.fL \times ... \times fHHA_n.fL.$

- fS ist die Menge der gesendeten Signale, wobei $fS \quad R \cup SHASig.Out.$ Die durch den SHA von inneren hybriden Automaten empfangenen Signale R werden im flachen Automaten auf die sendenden Signale der untergeordneten HHA abgebildet.

- fR ist die Menge der empfangenen Signale, wobei $fR \quad S \cup SHASig.In.$ Die von dem SHA an innere hybride Automaten gesendeten Signale S werden im flachen Automaten auf die empfangenen Signale der untergeordneten HHA abgebildet.

- fl_0 ist die Anfangslokation, welche dem Tupel $\langle fHHA_1.fl_0, ..., fHHA_n.fl_0\rangle$ entspricht.

- $fTF \subseteq fHHA_1.fTF \times ... \times fHHA_n.fTF$ ist die Menge der Endlokationen ohne Nachfolgelokation.

- $fFF \subseteq fHHA_1.fFF \cup fHHA_1.fTF \times ... \times fHHA_n.fFF \cup fHHA_n.fTF$ wobei $\exists i \; : \; l_i \in fHHA_i.fFF$ ist die Menge der Endlokationen mit Nachfolgelo-

kation.

- $fParam$ entspricht der Menge $Param$ und ist die Menge der nicht veränderbaren fest vorgegebenen Werte.

- $fClock \quad SClock \cup Clock \cup \bigcup_{i=1,...,n} fHHA_i.Clock$ ist die Menge von Uhren, welche ausgezeichnete kontinuierliche Variablen bilden.

- $fVar \quad SVar \cup Var \cup \bigcup_{i=1,...,n} fHHA_i.Var$ ist die Menge der Variablen.

- $fGuard \quad \bigcup_{\forall t \in f\delta} \langle GR_t,\ Cond_t \rangle$ mit $t = \langle l_t, \langle GR_t,\ Cond_t \rangle, \langle AS_t,\ Assign_t \rangle, l'_t \rangle$ ist die Menge der Wächter an Transitionen.

- $fAction \quad \bigcup_{\forall t \in f\delta} \langle AS_t,\ Assign_t \rangle$ mit $t = \langle l_t, \langle GR_t,\ Cond_t \rangle, \langle AS_t,\ Assign_t \rangle, l'_t \rangle$ ist die Menge der Aktionen an Transitionen.

- $f\delta \subseteq \{t = \langle l_t, \langle GR_t,\ Cond_t \rangle, \langle AS_t,\ Assign_t \rangle, l'_t \rangle \mid l_t = \langle l_1, ..., l_n \rangle,$

  $l'_t = \langle l'_1, ..., l'_n \rangle$ mit $\forall i \in \{1, .., n\}$ gilt : $l_i,\ l'_i \in fHHA_i.fL$
  und
  $\exists i \in \{1, .., n\}$ mit $\langle l_i, \langle GR_i, Cond_i \rangle, \langle AS_i, Assign_i \rangle, l'_i \rangle \in fHHA_i.f\delta$,
  wofür gilt :
  $(\forall j \in \{1, .., n\}$ mit
  $j \neq i \wedge HHA_j \in$ transitive_closure$(\emptyset, HHA_i, GR_i, Cond_i,$
  $AS_i, Assign_i, Connect).HHAs$
  gilt :
  $\exists \langle l_j, \langle GR_j, Cond_j \rangle, \langle AS_j, Assign_j \rangle, l'_j \rangle \in fHHA_j.f\delta$ mit
  $(GR_j =$ transitive_closure$(\emptyset, HHA_i, GR_i,$
  $Cond_i, AS_i, Assign_i,$
  $Connect).GR[HHA_j]) \wedge$
  $(AS_j =$ transitive_closure$(\emptyset, HHA_i, GR_i,$
  $Cond_i, AS_i, Assign_i,$
  $Connect).AS[HHA_j]))$
  und
  $(\forall j \in \{1, .., n\}$ mit
  $j \neq i \wedge HHA_j \notin$ transitive_closure$(\emptyset, HHA_i, GR_i, Cond_i,$
  $AS_i, Assign_i, Connect).HHAs$

gilt :
$l_j = l'_j \ \wedge \ l_j \in fHHA_j.fL$)
und
(GR_t transitive_closure($\emptyset, HHA_i, GR_i, Cond_i,$
$AS_i, Assign_i, Connect).GR \wedge$
AS_t transitive_closure($\emptyset, HHA_i, GR_i, Cond_i,$
$AS_i, Assign_i, Connect).AS$))
und wenn
$l_i = fHHA_i.fl_0 \ \wedge \ \forall j = 1, .., n \ \wedge \ j \neq i \ \wedge \ l_j = fHHA_i.fl_0$
dann
(($Cond_t$ transitive_closure($\emptyset, HHA_i, GR_i,$
$Cond_i, AS_i, Assign_i,$
$Connect).Cond \wedge$
$Connect_0.Cond$) $\wedge$
($Assign_t$ transitive_closure($\emptyset, HHA_i, GR_i,$
$Cond_i, AS_i, Assign_i,$
$Connect).Assign \cup$
$Connect_0.Assign$))
sonst
(($Cond_t$ transitive_closure($\emptyset, HHA_i, GR_i,$
$Cond_i, AS_i, Assign_i,$
$Connect).Cond$) $\wedge$
($Assign_t$ transitive_closure($\emptyset, HHA_i, GR_i,$
$Cond_i, AS_i, Assign_i,$
$Connect).Assign$))
}

ist die Übergangsrelation. Die Tupel der gegebenen Menge werden als Kombination aller möglichen Übergänge der bereits flachen Unterautomaten gebildet. Dabei entfallen Kombinationen aufgrund des Synchronisationsprinzips. Zu jedem Übergang eines Automaten HHA_i werden alle synchronisierten Übergänge von Automaten HHA_j ermittelt, die der transitiven Hülle bezüglich des Überganges des Automaten HHA_i angehören. Für alle Automaten HHA_j, die nicht der transitiven Hülle des Automaten HHA_i angehören, wird ein bedingungs- und zuweisungsloser Übergang einer bestehenden Lokation des Automaten HHA_j zu sich selbst erzeugt. Dieser Übergang entspricht einem faulen Übergang aus der Definition 2.1.1 des hybriden Automaten. Die empfangenen und gesendeten Signale GR_t und AS_t entsprechen der Konkatenation aller empfangenen und gesendeten Signale der Transitionen, die an der Synchronisation beteiligt sind. Die Bedingung $Cond_t$ und Zuweisung $Assign_t$ über den sich synchronisierenden Automaten HHA_i, wobei gilt i ∈ 1,..,n, entsprechen:

– der in der transitiven Hülle errechneten Gesamtbedingung und -zuweisung

verbunden mit der Anfangsbedingung und -zuweisung des SHA, wenn sich alle Automaten HHA_i in ihren Anfangszuständen befinden bzw.

– nur der in der transitiven Hülle errechneten Gesamtbedingung und -zuweisung, wenn mindestens ein Automat bereits seinen Anfangszustand verlassen hat.

Die Übergangsrelation bildet eine echte Teilmenge der oben definierten Menge von Tupeln, wenn zeitliche Bedingungen die Erreichbarkeit von Zuständen und damit Ausführung zugehöriger Transitionen verhindern. Diese Fälle können bisher nur dynamisch über Verfahren wie die symbolischen Simulation aufgedeckt werden.

- $fAct$ ist die Funktion, welche jeder Lokation $\langle l_1, ..., l_n \rangle \in fL$ eine Menge von Aktivitäten zuordnet, wobei gilt:
$fAct(\langle l_1, ..., l_n \rangle) \quad fHHA_1.fAct(l_1) \cup ... \cup fHHA_n.fAct(l_n).$

- $fInv$ ist die Funktion, welche jeder Lokation $\langle l_1, ..., l_n \rangle \in fL$ eine Invariante zuordnet, wobei gilt:
$fInv(\langle l_1, ..., l_n \rangle) \quad fHHA_1.fInv(l_1) \wedge ... \wedge fHHA_n.fInv(l_n).$

Lemma 4.4.1
Jeder beliebige synchronisierend hybride Automat SHA kann in einen flachen Automaten $fSHA$ transformiert werden, das Synchronisationsprodukt des SHA bildet.

Im Gegensatz zu den hierarchisch hybriden Automaten sei hier kurz die Beweisidee angegeben, da kein Transformationsalgorithmus für die Ausführung in CLP geschaffen werden musste. Die flache Modulstruktur von constraintbasierten Prologimplementationen erlaubt die direkte Ausführung kompositioneller Strukturen zur Synchronisation, doch keine hierarchischen Strukturen, um direkt Verfeinerungsmechanismen zu realisieren. Die Beweisidee beruht auf einem konstruktiven Verfahren, bei welchem der Algorithmus der transitiven Hülle angewandt wird. Deklarationen von Schnittstellen für die Umbenennung und das Verstecken von Bezeichnern sowie lokale Deklarationen werden dabei aus Vereinfachungsgründen nicht berücksichtigt.

Beweisidee 4.4.1
Ein synchronisierend hybrider Automat SHA sei auf der Basis einer endlichen Menge von n Unterautomaten HHA_i mit i 1,...,n gegeben. Die Unterautomaten HHA_i sind bereits in flache Automaten $fHHA_i$ überführt worden. Während der Transformation ist folgende Schrittfolge zur Berechnung einer Fixpunktmenge L über den zu konstruierenden Lokationen abzuarbeiten:

fl_0 : $\langle fHHA_1.fl_0, ..., fHHA_n.fl_0 \rangle$ % Anfangslokation des $fSHA$
L_i : $\{fl_0\}$ % Menge untersuchter und neu konstruierter Lokationen des $fSHA$
L_{i+1} : $\{\}$ % Menge aller untersuchten Lokationen
T : $\{\}$ % Menge aller Transitionen

while $L_{i+1} \neq L_i$ **do**
{
  L_{diff} : L_i - L_{i+1};
  L_{i+1} : L_i;
  forall $\{fl\} \in L_{diff}$ **do**
  {
    forall $fHHA_i.fl$ Teillokation von $\{fl\}$ **do**
    {
      forall Transitionen t_i, die von $fHHA_i.fl$ ausgehen **do**
      {
        if jede an der Synchronisation beteiligte Transition t_j eines
          $fHHA_j \in$ transitive_closure($\{\}, \{fHHA_i\}, GR.t_i, Cond.t_i,$
          $AS.t_i, Assign.t_i, Connect$).HHAs, die mit der in fl zugehörigen
          Teillokation $fHHA_j.fl$ beginnt **then**
        {
          t_{new} : Kombination aller an der Synchronisation beteiligten
            Transitionen mit Menge fauler Transitionen für alle nicht an
            der Synchronisation beteiligten $fHHA_j$ mit $j \neq i$,
            wobei $\langle fHHA_j, \langle\{\}, true\rangle, \langle\{\}, \{\}\rangle, fHHA_j\rangle$ eine faule
            Transition von $fHHA_j.fl$ aus fl ist;
          if $t_{new} \notin T$ **then**
            T : $T \cup \{t_{new}\}$;
          fl_{new} : Kombination der Ziellokationen aller an der Synchronisation
            beteiligten Transitionen mit allen $fHHA_j.fl$ aus fl der nicht
            an der Synchronisation beteiligten $fHHA_j$, wobei $j \neq i$
          if $fl_{new} \notin L_i$ **then**
            L_i : $L_i \cup \{fl_{new}\}$;
        }
        if alle $fHHA_i.fl \in fl_{new}$ Endlokationen der $fHHA_i$ **then**
          Endlokation des $fSHA$: fl_{new};
      }
    }
  }
}

Dieser Algorithmus erreicht den Fixpunkt in der Menge L, da jeder hybride Automat endlich viele Lokationen besitzt, woraus endlich viele, abzählbare Kombinationen gebildet werden können. Die Menge L bildet gerade die Lokationen des Synchronisationsproduktes, die unter der Beachtung von Synchronisationsverbindungen zwischen den Unterautomaten erreicht werden können. Grundlegende Definitionen für hybride Systeme sind in [Hen96, BR01] zu finden. Diese Definitionen wurden auf unsere synchronisierend und hierarchisch hybriden Automaten, welche Besonderheiten im Synchronisations- und Spezifikationsverhalten aufweisen, angepasst. Die Einschränkung der Lokationsmenge auf

eine Teilmenge der Mengen der einzelnen Automaten und der Transitionen auf eine Teilmenge der Menge der Transitionen nach dem Synchronisationsprinzip in der formalen Definition ist hier neu eingeführt worden. Erst mit der symbolischen Simulation kann ohne Annahmen zur Wohldefiniertheit von SHA ermittelt werden, welche Kombinationen von Lokationen bzw. Transitionen wirklich erreicht bzw. ausgeführt werden können. Weiterhin beachtet die formale Definition das Schnittstellenprinzip, welche aufgrund der Modularisierung zur Wiederverwendung geschaffen wurde.

4.4.3 Beispiel einer Synchronisation

Aus dem Abschnitt 4.3.3 der Verfeinerung von Automaten ist der Ablauf einer Prüfung, wie in Abbildung 4.9 nochmals vereinfachend ohne komplexe Lokation dargestellt, bekannt. Die Signale an den Transitionen repräsentieren dabei zeitlich verbundene Eintragungen in einem Datenbanksystem. Werden die Signale empfangen, so stellen diese Signale manuelle Eintragungen aus der Umgebung dar. Gesendete Signale stehen für zeitgesteuerte automatische Einträge, die als Information an die Umgebung weitergeleitet werden.

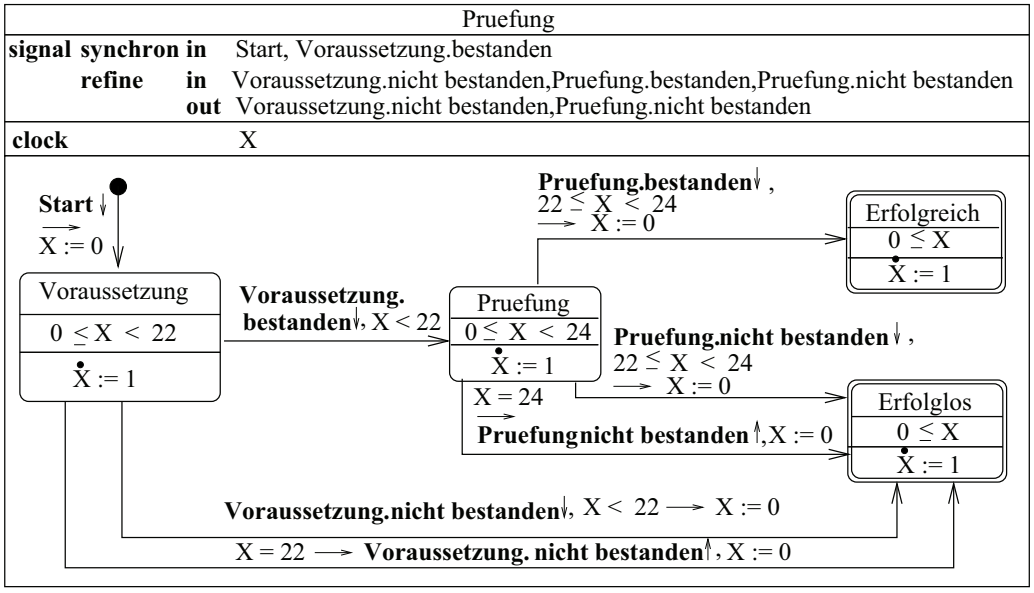


Abbildung 4.9: Automat eines Prüfungsablaufes

In der Umgebung sei ein Studienbüro für die manuellen Einträge in das Datenbanksystem verantwortlich, welches auch automatische Eintragungen als Zweitinformation registriert. Mit der Abbildung 4.10 ist der Ablauf der Verwaltungsarbeit in dem Büro beschrieben. Über ein Signal 'Start' aus der Umgebung wird der modellierte Prozess aktiviert. Der normale Studienverlauf ist in der Lokation 'Studium' beschrieben. Wird in einer Zeitspanne

von maximal 20 Zeiteinheiten (Monaten) eine Voraussetzung zur Prüfung bestanden, so erfolgt ein Übergang zur Lokation 'Pruefung'. Wird die Prüfung selbst in der Zeit von einschließlich 22 bis 24 Zeiteinheiten bestanden, so kann der Studienablauf normal fortgesetzt werden. Wurde die Prüfung in dieser Zeit nicht bestanden, so erfolgt durch das Studienbüro ein Vermerk in einer zweiten Kartei.

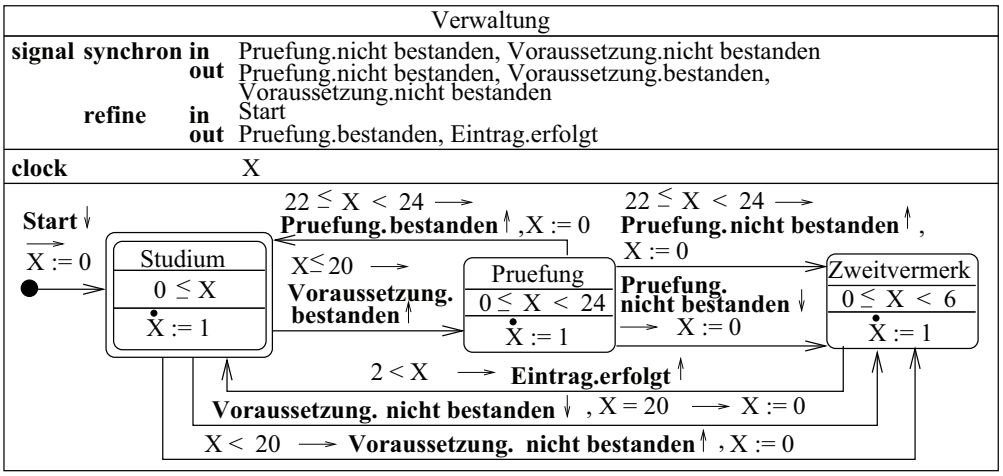


Abbildung 4.10: Automat des Ablaufes im Studienbüro

Dieser Vermerk beansprucht mindestens 2 Zeiteinheiten und bedarf im Höchstfall 6 Zeiteinheiten. Ist eine Prüfung im Zeitraum '$22 \leq X < 24$' nicht angegangen worden, so erfolgt gleichermaßen diese zweite Eintragung. Im Fall nicht bestandener bzw. nicht angegangener Voraussetzungen wird eine zweite Eintragung vorgenommen, nach der entsprechend der spezifizierten Zeitbedingungen innerhalb von 2 bis 6 Zeiteinheiten eine Rückkehr zur Endlokation 'Studium' erfolgt.
In der Abbildung 4.11 ist mithilfe eines synchronisierend hybriden Automaten das Zusammenwirken der Abläufe des Studienbüros und der Prüfung beschrieben. Eine Uhr 'S' symbolisiert den zeitlichen Verlauf für das gesamte System und wird mit der Wirksamkeit für alle Automaten an der Anfangsverbindung auf Null gesetzt.
Das 'Studienbuero' und die 'Einfache_Pruefung' bilden Instanzen der zuvor modellierten Automatentypen 'Verwaltung' und 'Pruefung'. Die Instanzen sind mit der Invariante '$0 \leq$ S' und der Aktivität '$\dot{S} :\ 0$' ausgestattet, die während der gesamten laufenden Prozesse der Instanzen gelten. Durch das Signal 'Beginn' aus der Umgebung werden beide Prozesse aktiviert. Alle manuellen Eintragungen, die von dem Studienbüro zur Information in der Datenbank festgehalten werden, sind durch die Signale:

Info_Vor_b	Information zur bestandenen Voraussetzung
Info_Vor_nb	Information zur nicht bestandenen Voraussetzung
Info_Pruef_b	Information zur bestandenen Prüfung
Info_Pruef_nb	Information zur nicht bestandenen Prüfung

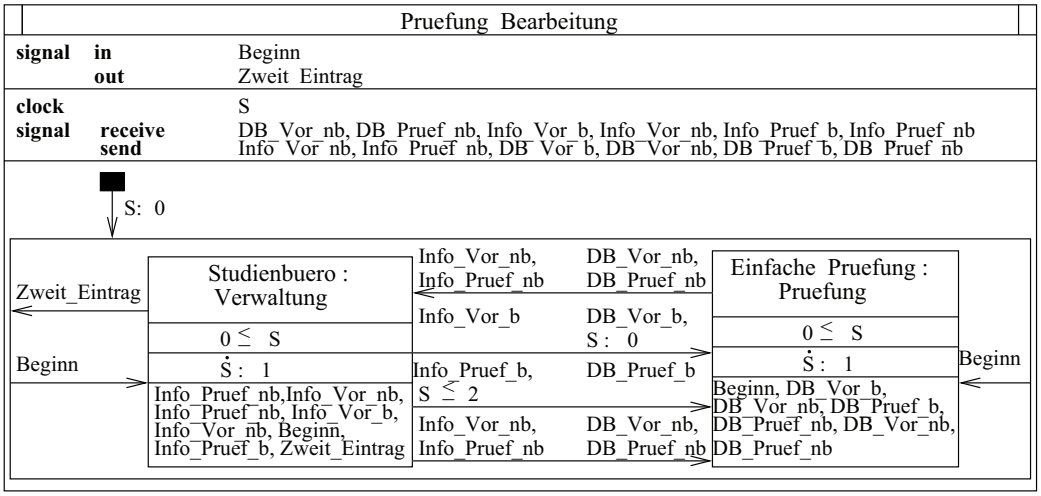


Abbildung 4.11: Synchronisation des Studienbüros mit dem Prüfungsablauf

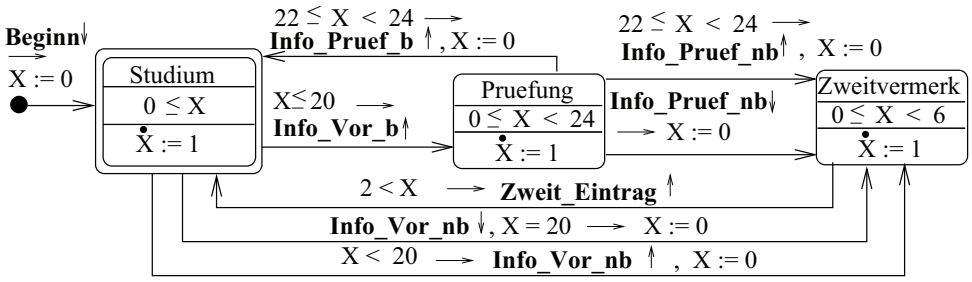


Abbildung 4.12: Ablauf im Studienbüro mit aktuellen Signalen

realisiert. Als 'DB_Vor_b', 'DB_Vor_nb', 'DB_Pruef_b' und 'DB_Pruef_nb' werden diese Signale in der Datenbank empfangen.
Umgekehrt werden automatische Eintragungen über nicht bestandene Voraussetzungen und Prüfungen durch die Signale 'DB_Vor_nb' und 'DB_Pruef_nb' von der Datenbank als Information 'Info_Vor_nb' und 'Info_Pruef_nb' an das Studienbüro gesandt. Die Information eines zweiten Eintrages wird in Form des Signals 'Zweit_Eintrag' vom Studienbüro an die Umgebung gesandt. Durch übergeordnete Verordnungen, die nicht direkt im Studienbüro bzw. in dem Prozess der einfachen Prüfung umgesetzt wurden, ist die Bedingung zu erfüllen:

> Eine Prüfung gilt nur dann als bestanden, wenn die Prüfung maximal 2 Zeiteinheiten nach dem Bestehen der Voraussetzung abgelegt wurde.

Diese Bedingung wird mithilfe der Uhr 'S' modelliert, wobei 'S' im Fall der bestandenen Voraussetzung auf Null gesetzt wird und im Fall einer laut Studienbüro bestandenen Prüfung mit der Synchronisationsbedingung '$S \leq 2$' verbunden wird.

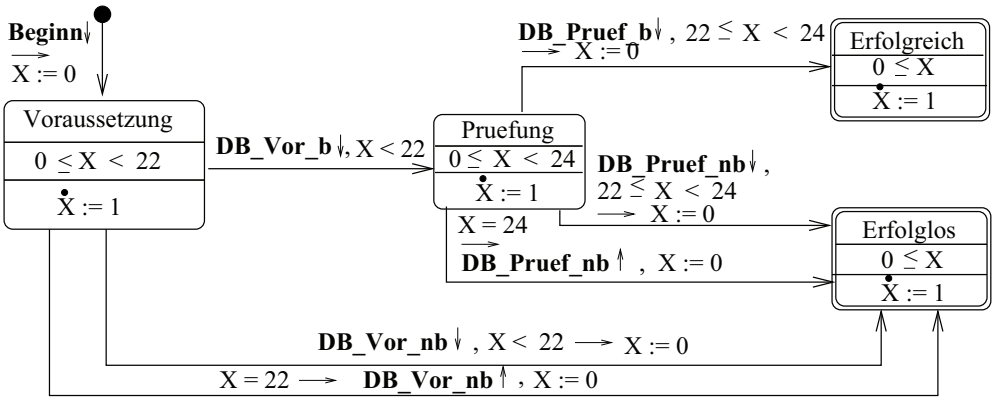


Abbildung 4.13: Ablauf der Prüfung mit aktuellen Signalen

In den Abbildungen 4.12 und 4.13 sind beide Verhaltensbeschreibungen des Studienbüros und der einfachen Prüfung zum leichteren Nachvollziehen des entstandenen Synchronisationsproduktes der Abbildung 4.14 mit den umbenannten Signalen enthalten. Unter dem Namen 'Pruefung_Bearbeitung' ist das Synchronisationsprodukt der Abbildung 4.14 mit der aus dem Verhalten abgeleiteten Schnittstelle und der lokalen Uhrendeklaration dargestellt.
Zur einfacheren Handhabung wurden die Uhren der Automaten 'Studienbuero' und 'Einfache_Pruefung' in 'A' und 'B' umbenannt. Die Abkürzung 'SB' steht für 'Studienbuero' und 'EP' für 'Einfache_Pruefung'. Viele Kombinationen von Lokationen entfallen durch die mehrfache Synchronisation zwischen dem 'Studienbuero' und der 'Einfachen_Pruefung'. Zum Beispiel kann sich der gesamte Prozess nicht gleichzeitig in der Lokation 'SB.Studium' und 'EP.Pruefung' befinden. Durch Verbindungen von Invarianten und Transitionsbedingungen kann der Aufenthalt in einer Lokation und die Ausführung einer Transition zeitlich eingeschränkt werden. Die Lokation 'SB.Studium' muss hier z.B. entsprechend der hinzugefügten Bedingung '$0 \leq B < 22$' auch innerhalb von 22 Zeiteinheiten verlassen werden. Im Studienbüro wurde durch '$A < 20$' bereits eine härtere Restriktion bezüglich des Bestehens der Voraussetzung umgesetzt, die z.B. aus einer neuen Durchführungsordnung stammt. Diese Einschränkung wurde für die 'Einfache_Pruefung' noch nicht geändert. Im Synchronisationsprodukt ist deutlich zu erkennen, dass die neue Einschränkung Auswirkungen auf alle Prozesse hat, in welchen das Bestehen bzw. Nichtbestehen über 'Vor_b' bzw. 'Vor_nb' eine Rolle spielt. Der Student kann eine Voraussetzung nur noch innerhalb von 20 Zeiteinheiten ablegen.
Obgleich das Studienbüro durch die Modellierung von Zyklen die Möglichkeit besitzt, den Arbeitsablauf nach Erreichen der Endlokation 'Studium' mehrfach zu durchlaufen, zeigt das Synchronisationsprodukt deutlich, dass mit den Endlokationen 'Erfolgreich' und Erfolglos' der einfachen Prüfung nach einem Durchlauf ein Abschluss für den gesamten Prozess des synchronisierend hybriden Automaten 'Pruefung_Bearbeitung' gegeben ist. In der praktischen Anwendung ist dieses Ergebnis nicht zufriedenstellend. Somit wurde

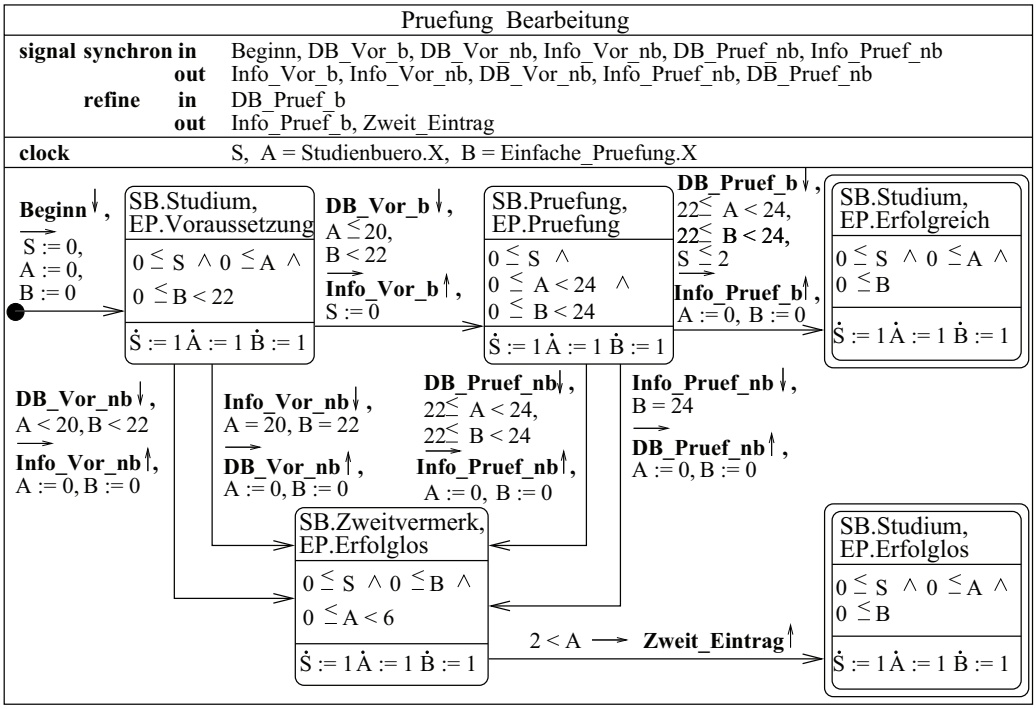


Abbildung 4.14: Produktautomat der Synchronisation

im Anhang E auf eine Lösungsmöglichkeit in unseren Sprachen eingegangen.

Kapitel 5

Verfahren und Werkzeuge zum Nachweis von Eigenschaften

Die Analyse hybrider Systeme erfolgt auf der Grundlage unterschiedlicher Verfahren, mit welchen ein Modell untersucht werden kann. Die Art der Eigenschaften besitzt eine besondere Bedeutung bei der Auswahl und Kombination der eingesetzten Verfahren. Grundlegend können die Verfahren in Simulation und Verifikation eingeteilt werden. Die klassische Simulation [Möl92, ZPK00] kann durch folgende Begriffsbildung beschrieben werden:

Begriff 5.0.2
Die **Simulation** bildet eine Problemlösungsmethode, bei der durch Experimente auf Modellen Aussagen über das Verhalten der durch die Modelle dargestellten Systeme gewonnen werden können. Systeme, deren Berechnung aufgrund nicht existierender oder sehr komplizierter mathematischer Systembeschreibungen nicht möglich ist und an denen in der Realität keine Experimente möglich sind, bilden den Gegenstand der Anwendung von Simulationen.

Erkenntnisse zu Eigenschaften von Verhaltensweisen natürlicher Prozesse der Biologie, Chemie und Physik sind Beispiele solcher Systeme. Die große Vielgestaltigkeit der genannten Systeme stellt durch die Komplexität der sich daraus ergebenden Zustandsräume für die Ausführung der formalen Verifikation unter Verwendung heutiger Hardware- und Softwarekomponenten oft noch ein Problem dar. Deshalb werden Verifikationstechniken häufig im Bereich vom Menschen selbst erschaffener Systeme, wie automatischer Überwachungs- und Steuerungssysteme [VPV06], eingesetzt.

Begriff 5.0.3
Die **Verifikation** bildet einen formalen Nachweis von Eigenschaften für Systeme, denen eine präzise Semantikdefinition zugrundeliegt und dient dem Beweis der Korrektheit dieser Systeme. Dabei heißt ein System korrekt, wenn es der vorgegebenen Spezifikation der Eigenschaften entspricht.

In Fällen bekannter, vorhersagbarer und sich wiederholender Verhaltensweisen können Abstraktions- und Reduktionstechniken zur Einschränkung des Zustandraumes angewandt [JKSC05] werden. Zur Verifikation werden Eigenschaften als Anforderungen definiert, die genau festlegen, wonach gesucht wird. Dabei können die Eigenschaften in vorgenommenen Klassifikationen eingeordnet werden [BBF+99]. Verifikationstechniken lassen sich in deduktive und automatische Vorgehensweisen einteilen.
Die *deduktive Verifikation* erfolgt auf der Grundlage mathematischer Beweisregeln mit Theorembeweisern [ORSSC98, NPW02, Mit07, PQ08, NT08]. Mathematische Beweise können in Modellen mit einer unendlichen Anzahl an Zuständen, wie für parallele Programme mit Datenstrukturen über unendlichen Domänen und mit unendlichen Kontrollstrukturen [Nie02], durchgeführt werden. Interaktionen, wie z.B. die Bestimmung einer Invariante durch einen Experten, bilden eine notwendige Voraussetzung für Theorembeweiser. Im Fall eines gescheiterten automatischen Beweises kann ein manueller Beweis angeschlossen werden.
Durch die Einführung von Zyklen, Abstraktionen von detaillierten Betrachtungen und konkrete Instantiierung von Parametern können Zustandsräume für viele praktische Probleme auf eine endliche Anzahl an Zuständen reduziert werden, wodurch eine *automatische Verifikation* wie das Model Checking ausreichend ist. Interaktionen sind für automatische Verifikationstechniken nicht notwendig. Im Fall eines gescheiterten Beweises kann ein Gegenbeispiel als Gegenbeweis automatisch nachvollzogen werden. Ein umfassender Überblick über Arten der Verifikation von Software und deren Abgrenzung bezüglich ihrer Vor- bzw. Nachteile ist in [DHR+07] gegeben. Zur Nutzung der Vorteile verschiedener Verifikationstechniken in einer Anwendung wurden in [MN95, SUM96, Ber02, dMRS02, GNRZ07] deduktive und automatische Methoden kombiniert. Für hybride Systeme sind mit HyTech [HHWT95], Kronos[Yov97], Uppaal [LPY97], Checkmate [SRKC00], d/dt [ADM02], SpeeDI [APSY02], PHAVer[Fre05], Sphin [SCR06] und Hsolver [RS07] Werkzeuge entstanden, mit deren Hilfe die Kombination aus kontinuierlichem und diskretem Verhalten untersucht werden kann.
Die *symbolische Simulation* stellt eine Art der automatischen Verifikation, die mit den Mitteln einer Simulation durchgeführt wird, dar. Im Folgenden wird die Symbolische Simulation in Abgrenzung zur klassischen Simulation und zum Model Checking genauer betrachtet.

5.1 Klassische Simulation

Im Vorfeld der Simulation hybrider Systeme wird die Simulation kontinuierlicher Systeme und die Simulation diskreter Systeme betrachtet. Die Ideen der grundlegenden Vorgehensweisen einzelner Simulationsarten sollen dabei dem Verständnis der Simulation in Systemen mit sowohl kontinuierlichen als auch diskreten Komponenten dienen.

5.1.1 Simulation kontinuierlicher Systeme

Kontinuierliche Systeme werden über Differentialgleichungen beschrieben [ZPK00]. Da zu jedem beliebigen Punkt der Zeitachse ein Zustand und ein Eingabewert existiert, kann eine Aufeinanderfolge von Zuständen nicht direkt dargestellt werden. Nur die Veränderungsraten kontinuierlicher Zustandsvariablen können mithilfe von Ableitungsfunktionen spezifiziert werden. Ein elementares kontinuierliches System ist dabei ein einfacher Integrator, der eine Eingabevariable x und eine Ausgabevariable y besitzt. Der Integrator verhält sich wie ein Speicher unendlicher Kapazität über die Zeit t, wobei alle positiven Eingaben vom Integrator akkumuliert und negative Eingabewerte abgezogen werden. Durch die Variable y wird der gegenwärtige Inhalt des Integrators ausgegeben. Der gegenwärtige Inhalt selbst ist durch die Zustandsvariable q gekennzeichnet. Die Rate der gegenwärtigen Inhaltsveränderung wird durch x repräsentiert. Die Differentialgleichung

$dq(t)/dt = x(t)$

stellt die Beziehung mathematisch dar. Weiterhin gilt, dass die Ausgabe y dem gegenwärtigen Zustand q zu einem Zeitpunkt t entspricht:

$y(t) = q(t)$.

Im Allgemeinen existiert eine Menge von Zustandsvariablen und somit auch eine Menge von Differentialgleichungen, die als Differentialgleichungssystem in Integratorblöcken simuliert werden.

5.1.2 Simulation diskreter Systeme

Die Beschreibung der Simulation diskreter Systeme erfolgt in Anlehnung an [CL99]. In diskreten Systemen ist der Zustandsraum durch eine diskrete Menge wie {0,1,2,...} beschrieben und Zustandsänderungen erfolgen zu bestimmten Zeitpunkten durch Transitionen, die ohne Zeitverzug ausgeführt werden. Mit jeder Transition kann ein Ereignis verbunden sein. Existiert für ein betrachtetes diskretes System eine Uhr zum Messen der Zeit, so können bezüglich der Ereignisse und der Zeit zwei Möglichkeiten bestehen:

1. Zu jedem Tick der Uhr tritt ein Ereignis auf. Wenn kein Ereignis auftritt, so existiert ein 'Null-Ereignis', das in Erscheinung tritt, wobei kein Zustandswechsel erfolgt.

2. Zu verschiedenen Zeitpunkten, die nicht notwendigerweise im Voraus bekannt sein und mit denen nicht notwendigerweise Uhren-Ticks zusammenfallen müssen, erscheint irgendein Ereignis.

Der Unterschied zwischen 1. und 2. ist:

- In 1. werden Zustandstransitionen durch die Uhr synchronisiert: die Uhr tickt, ein Ereignis oder ein Null-Ereignis erscheint, der Zustand ändert sich und der Prozess wird wiederholt. Die Uhr ist allein für mögliche Zustandsübergänge verantwortlich.

- In 2. definiert jedes Ereignis den Prozess, durch welchen der Zeitpunkt des Auftretens des Ereignisses bestimmt wird. Zustandsübergänge sind das Ergebnis der Kombination dieser asynchronen und nebenläufigen Ereignisprozesse. Diese Prozesse können voneinander abhängen.

Dieser Unterschied führt zu den Begriffen **zeit-gesteuert** und **ereignis-gesteuert**. Während kontinuierliche Systeme von Natur aus zeit-gesteuert sind, können diskrete Zustandssysteme sowohl zeit-gesteuert als auch ereignis-gesteuert auftreten. Sind die Zustandsübergänge durch eine Uhr synchronisiert, so handelt es sich um zeit-gesteuerte, diskrete Zustandssysteme. Erscheinen die Zustandsübergänge asynchron in Abhängigkeit von auftretenden Ereignissen, so handelt es sich um ereignis-gesteuerte, diskrete Zustandssysteme. Auf diese Art und Weise kann ein diskret-ereignisorientiertes System wie folgt beschrieben werden.

Begriff 5.1.1
Ein **Diskret-Ereignisorientiertes System** (DES) ist ein ereignis-gesteuertes, diskretes Zustandssystem, wobei die Zustandsentwicklung des Systems vollständig von dem Auftreten asynchroner, diskreter Ereignisse über der Zeit abhängig ist.

Da diskret-ereignisorientierte Systeme oftmals zu komplex sind, um analytische Lösungen für explizite, mathematische Aussagen über den Zustand von Größen der DESs liefern zu können, wird eine numerisch ausgeführte Simulation, die **diskret-ereignisorientierte Simulation**, zur Untersuchung der Systeme genutzt.

5.1.3 Simulation hybrider Systeme

Als klassische Simulation hybrider Systeme wird hier die Reproduktion des Verhaltens dieser Systeme basierend auf einer Menge von konkreten Läufen der Definition 2.2.1 verstanden. Jede ausgeführte Transition eines Laufes wird zu einem festen Zeitpunkt mit einer konkreten Belegung der Werte aller Variablen des hybriden Systems betrachtet. Durch die Vereinigung von Größen eines diskret arbeitenden Automatisierungssystems mit Größen einer sich kontinuierlich ändernden Umgebung existieren in hybriden Systemen unendlich viele konkrete Läufe. Die Untersuchung einer genügend großen Menge solcher Läufe ist das Ziel der klassischen Simulation hybrider Systeme. Als Ergebnis können Aussagen über Eigenschaften der Systeme getroffen werden, die durch die Untersuchung mit einer hohen Wahrscheinlichkeit belegt sind. Für die Simulation hybrider Systeme sind Werkzeuge wie BaSiP [WFSE96], OmSim [BM97], COSIMO [Kot97], Smile [EKRNS00], ABACUSSII [TCB02], HyBrSim [Mos02], AnyLogic [BKK02], Ptolemy II [BLL$^+$05] und MOSILAB [ENNG$^+$05, NGEN$^+$06] entstanden. Eine Übersicht über Möglichkeiten der Untersuchung von Simulationserscheinungen wie:

- Aufdecken und Lokalisieren von Zuständen und Zeiten,

- Erkennen von Chattering, dem Verhalten für unendlich viele diskrete Schritte innerhalb einer kleinen Zeitspanne bzw.

- Zenoness, der Konvergenz von Schrittfolgen in einer Zeitspanne

mit den angegebenen Werkzeugen sind in [Mos99] enthalten.

5.2 Model Checking

Mit Einführung temporaler Logiken wurde beobachtet, dass temporale Eigenschaften für endliche Transitionssysteme automatisch nachweisbar sind [CE82, QS82]. Eine Prozedur, welche überprüft, ob ein Transitionssystem ein Modell für eine temporal-logische Formel ist, wird als *Model Checker* bezeichnet. Der Begriff des Model Checking [CGP99, HR00, Mer01, BK08] kann folgendermaßen erklärt werden.

Begriff 5.2.1
Model Checking ist eine Art der automatischen Verifikation, bei welcher die Frage beantwortet wird, ob ein nebenläufiges, zustandsbasiertes Modell eine Eigenschaft beschrieben in einer Temporallogik erfüllt.

Die Komplexität der Model Checking Prozeduren hängt dabei von der gewählten Temporallogik [BMN00] zur Beschreibung der Eigenschaft und der zugrundeliegenden Ausführungstechnik, wie diese in den folgenden Abschnitten beschrieben sind, ab. Für Temporallogiken, die um den Aspekt der Zeit erweitert wurden, wie TCTL [ACD90, ACD93] und TPTL [AH89, AH92] wurde gezeigt, dass Fragen nach der Gültigkeit von Formeln dieser Logiken in Bezug auf den unendlich dichten Zeitbereich unentscheidbar sind. Doch konnte für einen vollständigen Teil von TPTL, der der Theorie von zeitlichen Zustandsfolgen [AH93] entspricht und somit entscheidbar ist, eine tableaubasierte Entscheidungsprozedur und ein Algorithmus zum Model Checking entwickelt werden. In [BL95, Bou08] ist zur Entscheidbarkeit und zum Model Checking temporaler Logiken ein umfassender Überblick zu finden. In den nächsten Abschnitten wird kurz auf die Entwicklung verschiedener Arten des Model Checking eingegangen.

5.2.1 Symbolisches Model Checking

Die ersten Model Checking Prozeduren für temporal-logische Formeln wurden mithilfe von iterativen Fixpunktberechnungen über der Menge von Zuständen, in welcher die Formel gültig sein soll, ausgeführt. Doch dies erforderte den Aufbau eines endlichen Transitionsmodells für das zu untersuchende System, was jedoch mit einem Zustandsexplosionsproblem [CG87] verbunden war. Um dieses Problem in einigen Fällen zu vermeiden, wurde in [McM93] das *Symbolische Model Checking* entwickelt. Das symbolische Model

Checking ist eine automatische Methode, bei welcher Boolsche Formeln, die hier implizit die Transitionsmodelle repräsentieren, manipuliert werden. Als Repräsentation zur Reduktion der Formeln wurden geordnete binäre Entscheidungsdiagramme (OBDDs, Ordered Binary Decision Diagrams) [Bry86] verwendet. Auf dieser Basis bleiben Formeln vergleichbar und können unter heuristisch günstigen Bedingungen die Reduktion einer großen Anzahl an Zuständen ermöglichen [BCM$^+$90, MLAH99].
Das Werkzeug SMV wurde in [McM93] zum symbolischen Model Checking über Kripke-Modellen entwickelt. Ein weiteres Werkzeug ist Rabbit [BLN03], welches zum Model Checking hybrider Systeme auf der Basis von BDDs entstand. Das symbolische Model Checking wurde auch in HyTech [HNSY94, AHH96] realisiert, wobei Eigenschaften der Temporallogik CTL als endlich angenäherte Fixpunkte in einer einfachen entscheidbaren Theorie nachgewiesen werden.

5.2.2 Bounded Model Checking

Bounded Model Checking (BMC) lässt sich eng mit unserer symbolischen Simulation verbinden, wie in Abschnitt 5.3.4 genauer beschrieben ist. *Bounded Model Checking* wurde erstmals in dem Artikel [BCCZ99] vorgestellt. Die grundlegende Idee des BMC beruht auf der Suche nach Gegenbeispielen für den Nachweis von Eigenschaften, deren Länge eine Grenze, die durch eine natürliche Zahl k gegeben ist, nicht überschreiten. Wenn kein Fehler gefunden wurde, so wird k entweder bis zum Finden eines Fehlers, d.h. das Problem ist unlösbar, oder bis zu einer vorher bekannten Obergrenze schrittweise inkrementiert. Eine bekannte Obergrenze wird als *Durchmesser eines Entwurfes* bezeichnet. Ein Entwurf besteht dabei aus einem gegebenen Modell, einer zu prüfenden Eigenschaft und einem Übersetzungsschemata in eine SAT-Formel. Der Durchmesser d gibt die Länge eines Zyklus an, wofür gilt, wenn bis zur Obergrenze d kein Fehler auftratt, so kann auf die Fehlerfreiheit des gesamten Verhaltens des vorliegenden Modells geschlussfolgert werden. Bounded Model Checking wird nicht mehr auf der Basis von BDDs durchgeführt, sondern auf der Basis von Formeln, die durch die Nutzung von Entscheidungsprozeduren für SAT-Probleme gelöst werden [BCC$^+$03]. Die SAT-Technik weist entsprechend [CBRZ01] folgende Vorteile auf:

- benötigt keinen exponentiellen Zustandsraum,
- kann somit wesentlich größere Entwürfe als das explizite Model Checking auf BDD Basis verifizieren (BDD Verfahren verbrauchen zusätzlich über die Breitensuche viel Speicherplatz.),
- kann zu einem schnellen Nachweis führen, da Suchverfahren in beliebiger Ordnung möglich sind,
- ist in der Lage, Pfade minimaler Länge zu finden,

- benötigen im Allgemeinen weniger manuelle Änderungen als BDDs, da Standardfälle von Heuristiken über Teilungsmechanismen ausreichend sind.

Nachteile der Anwendung des Bounded Model Checking sind in folgenden Punkten zu sehen:

- Obgleich BMC erweiterbar ist, wurde die Methode bisher nur auf Spezifikationen angewandt, in denen sich Fixpunktoperationen leicht vermeiden lassen.

- Bounded Model Checking ist im Allgemeinen nicht vollständig, d.h. eine Garantie für eine endgültige Aussage über die Erfüllbarkeit einer Eigenschaft kann nicht immer gegeben werden.

- Da die Länge der zu prüfenden Formel mit jeder Transition wächst, werden sehr lange Nachweise und Gegenbeispiele und allgemein die Untersuchung aller möglichen Pfade für einen Großteil von Modellen verhindert.

Trotz der Nachteile bietet BMC eine ausgleichende Kombinationsmöglichkeit zu anderen Verifikationsverfahren in Fällen sich ständig wiederholender Verhaltensweisen.
Das Werkzeug NuSMV [CGP+02] und dessen Nachfolger NuSMV2 [CCG+02] kombinieren BDD- und SAT-basierte Techniken, um unterschiedliche Klassen von Problemen zu lösen. Mit [ACKS02] und [Sor02] sind Untersuchungen des BMC für Zeitsysteme und Zeitautomaten entstanden. Bounded Model Checking in [ACKS02] wird mittels MathSAT, einem Löser mit integrierten Entscheidungsprozeduren für lineare mathematische Constraints in SAT-Techniken, ausgeführt. In [Sor02] wird gezeigt, dass BMC für Zeitautomaten vollständig ist und kleinste bzw. oberste Schranken für die gesetzte Grenze k zum Berechnen von Gegenbeispielen angegeben. Für lineare hybride Systeme wurde mit HySAT ein Bounded Model Checker in [FH05, FH07] vorgestellt, der in [HEFT08] für nichtlineare Systeme erweitert wurde. Auf der Grundlage eines Pseudo-Boolschen SAT-Lösers über der klassischen David-Putnam-Loveland-Logemann (DPLL) Prozedur und linearer Programmierung [WW99] wird das Bounded Model Checking dort mit Techniken wie konfliktgetriebenes Lernen und Beschleunigen der Beweissuche über:

- ausgereifte Heuristiken zur Auswahl von Zuweisungen einzelner Entscheidungsschritte,

- Hinzufügen verschiedener algorithmischer Verfeinerungen,

- nicht-chronologisches Backtracking,

- zufällige Neustarts und

- verzögerte Klauselauswertung

erweitert. Zur Anwendung des Bounded Model Checking in verschiedenen Bereichen wie z.B. hybride Systeme, digitale Schaltungen, Planung und heuristische Suche und der Verbindung des BMC mit verschiedenen Verifikationsverfahren wurde mit [BBE$^+$04] eine Grundlage geschaffen. Auf unendliche Zustandssysteme wurde BMC in [SS07] angewandt und in folgendem Kontext untersucht:

- Anstelle der Aussagenlogik wurden aussagekräftigere Logiken bezüglich unendlicher Zustandssysteme verwendet, welche jedoch immer noch entscheidbar sind.
- Statt der Auflösung der Hierarchien durch Unterformeln in temporal-logischen Formeln und deren schrittweisen Abarbeitung nach dieser Auflösung werden temporallogische Formeln unter Berücksichtigung ihrer Hierarchisierung in ω-Automaten überführt. Dabei bleiben die ursprünglichen Sicherheits- und Lebendigkeitseigenschaften erhalten.
- Es werden globale und lokale Model Checking Prozeduren verwendet, um verschiedenen Typen von Spezifikationen gerecht zu werden.

In [CKOS04] erfolgte eine genauere Betrachtung der Vollständigkeit und Komplexität des Bounded Model Checking. Dabei werden Aussagen zur Vollständigkeit und Komplexität bezüglich eines Schwellwertes der Vollständigkeit getroffen. Dieser Schwellwert besagt, wenn es kein Gegenbeispiel zu einer Eigenschaft in einem gegebenen Modell bis zur Pfadlänge dieses Schwellwertes gibt, dann soll das Modell die Eigenschaft erfüllen. Für den Fall, dass solch ein Schwellwert gefunden werden kann, kann die Vollständigkeit des BMC nachgewiesen werden. Mit Hilfe von Büchi-Automaten wird die Berechnung von Vollständigkeitsschwellwerten für allgemeine Eigenschaften der LTL-Logik beschrieben. Für standardmäßig SAT-basiertes BMC wird gezeigt, dass die Untersuchung von Graphen mit doppelter Exponentialität bezüglich der besuchten Zustände erfolgt, wogegen LTL Model Checking mit einfacher exponentieller Zeit auskommt. Der Grund für diesen Unterschied wird in der Art von Durchläufen gesehen. Während sich LTL Model Checking auf BDD-Basis bereits erreichte Zustände merkt, wurden beim ursprünglichen SAT-basierten BMC im schlechtesten Fall alle möglichen Pfade durchlaufen, da kein Gedächtnis der bereits besuchten Zustände vorhanden war. Zur Lösung dieses Problems wird in [CKOS04] ein entsprechender SAT-basierter BMC-Algorithmus angegeben.

5.2.3 Model Checking in CLP

Schon frühzeitig wurde die constraint-logische Programmierung (CLP) als Paradigma zur Verifikation von Echtzeitsystemen [PG97] und hybriden Systmen [Mis95, NC94, Rie95, RU95] genutzt. In [DP99] wurde erstmalig die Ausführung des Model Checking unendlicher Zustandssysteme in CLP beschrieben. Dabei wurde die Umsetzung nebenläufiger Systeme in CLP gezeigt und eine Methode zur Verifikation von Sicherheits- und Lebendigkeitseigenschaften angegeben. Detailliertere Beschreibungen der Semantik für die

Übersetzung des Modells in CLP und der Ausführung des Model Checking wurden später in [DP01] veröffentlicht. Im Zusammenhang mit dem Software Model Checking stellt [Fla03] eine Methode zur Überprüfung traditioneller, imperativer Programme in CLP vor, wobei eine semantikerhaltende Übersetzung imperativer Sprachen mit veränderlichen, über Halden belegbare Datenstrukturen und mit rekursiven Prozeduren in CLP erfolgt. Die Arbeit von [PR07] beschäftigt sich mit der Entwicklung des Werkzeuges ARMC. Die Fixpunktdefinitionen der Laufzeiteigenschaften von CLP-Programmen zum Model Checking wurden bisher mit der Fixpunktsemantik constraint-logischer Programme in Zusammenhang gebracht. Dieser Zusammenhang ist für das Verständnis der Ausführung des Model Checking auf rein operationalem Weg in CLP sinnvoll, doch um ein logisches Lesen der CLP-Programme in Bezug auf spezifische Techniken des Model Checking wie Abstraktion und anschließende Verfeinerung zu erleichtern, werden CLP-Programme in ARMC entsprechend dieser Techniken mit problemspezifischen Sprachelementen ausgestattet.

5.3 Symbolische Simulation

Die symbolische Simulation kann wie in folgender Begriffsbildung beschrieben werden:

Begriff 5.3.1
Die **symbolische Simulation** ist eine Art der automatischen Verifikation, bei welcher die Frage beantwortet wird, ob ein nebenläufiges, zustandsbasiertes Modell eine Eigenschaft beschrieben als symbolische Trajektorie erfüllt.

Der Begriff der *Trajektorie* entspricht dabei dem Begriff des Laufes der Definition 2.2.1 und dementsprechend der Begriff der *symbolischen Trajektorie* dem Begriff des symbolischen Laufes der Definition 2.2.2. Während die Simulation mit konkreten Trajektorien auf konkreten Werten von Variablen arbeitet, wird die symbolische Simulation über *symbolische Trajektorien* mit symbolischen Werten von Variablen ausgeführt. Ein symbolischer Wert kann unendlich viele konkrete Werte verkörpern und durch Bedingungen auf ein gegebenes Intervall eingeschränkt sein. Unterschiedliche Arten der symbolischen Simulation ergeben sich aus unterschiedlichen:

1. Eigenschaften, die nachgewiesen werden sollen,
2. Arten von zustandsbasierten Modellen, die den Systemen zugrundeliegen und
3. Ausführungsmechanismen.

In [Mau96] werden dazu 4 Eigenschaften katagorisiert:

- augenblicklich, logische Eigenschaften (Aussagenlogik) zur Überprüfung von Tautologien,

- augenblicklich, numerische Eigenschaften, welche mittels abstrakter Interpretation über polyedrischen Wertebereichen überprüft werden,

- temporal-logische Eigenschaften, die über die Erreichbarkeit von Zuständen überprüft werden,

- temporal-gemischte numerische/logische Eigenschaften, wobei auf abstrakten Wertebereichen (Polyedern) eine obere Annäherung der Menge erreichbarer Zustände durch die Konvergenz einer Fixpunktberechnung ermittelt wird.

Im Gegensatz zur Arbeit von [Mau96], welcher die symbolische Simulation auf abstrakt interpretierten Automaten synchroner Programme ausführt und somit numerische Eigenschaften z.B bei der Vereinfachung von Bedingungen beachten muss, findet die Ausführung unserer Automaten in bereits vorliegenden CLP Implementationen statt. Auf diese Art und Weise beschäftigt sich unser Ansatz mit den temporal-logischen Eigenschaften, ohne explizit auf numerische Aspekte einzugehen, die durch die Implementation bereits gegeben sind. Hierzu wird in unserer Arbeit dem im Abschnitt 5.2.3 zum Werkzeug ARMC angesprochene Ansatz entsprochen, wobei die Ausführung der symbolischen Simulation und somit die Abarbeitung von Wörtern im Zusammenhang mit der operationalen Semantik von CLP betrachtet wird.
Die symbolische Simulation wurde unter anderem für Modelle, wie:

- Hybride Automaten [NC94, RU95, Rie95],

- Hybride Systeme basierend auf kontinuierlichen Aktionssystemen [BSW02], bestehend aus:

  – einer endlichen Menge von Attributen, deren Wertebereiche sich über zeitlich diskret und kontinuierlich ändernde Werte erstrecken und den Zustand des Systems darstellen, zusammen mit

  – einer endlichen Menge an Aktionen, die auf die Attribute wirken,

- DEVS (Discrete Event System Specification) [LC05],

- nebenläufige Zeitautomaten [Wan07].

realisiert. Symbolische Simulatoren wurden für:

- CLP [NC94, Mis95, RU95, MT99, Gar02, HW04],

- Simulink/Stateflow, Matlab zur Modellierung [Tiw02],

- Mathematica [BSW02],

- Scheme (Lisp Dialekt), Visual C++ Umgebung [LC05],

implementiert. In [Tiw02] ist die Simulation des kontinuierlichen Verhaltens verbunden mit diskreten Zustandsänderungen von Interesse, wobei die Beschreibung des Zustandsmodells in Stateflow auf der Basis kommunizierender Kellerspeicher erfolgt. In Ansätzen aus [NC94, Mis95] und [LC05] werden durch die symbolische Simulation Folgen von diskreten Ereignissen verbunden mit symbolischen Variablen überprüft. Die Arbeit von [HW04, Wit04, WH06] ist mit unserer Ausführung stark vergleichbar, da hier in CLIP, einem System basierend auf CLP(F), die Modellierung und Analyse hybrider Systeme mit Hilfe von Intervallarithmetik und nichtlinearen Constraints durchgeführt wird. Unser Ansatz lehnt sich wie zuvor [RU95, Rie95] an die Vorgehensweise von [NC94, Mis95] an. Die Anwendung besteht dabei in dem Nachweis zeit- und sicherheitskritischer Eigenschaften an den Übergängen und Synchronisationsverbindungen hybrider Automaten. Die *symbolische Simulation* von Folgen diskreter Ereignisse verbunden mit symbolischen Variablen wird in dieser Arbeit entsprechend [RU95] als *Prozess der Akzeptanz von Wörtern* in hybriden Automaten betrachtet. Auf diese Art und Weise werden das Verhalten und Eigenschaften hybrider Automaten auf der Grundlage formaler Sprachen beschrieben, wodurch in weiteren Arbeiten theoretische Erkenntnisse für die Untersuchung dieser Automaten genutzt werden können. Kontinuierliche Prozesse der Lokationen in den Automaten werden als korrekt angenommen bzw. müssen in Verbindung mit Simulationen für kontinuierliche Systeme als Voraussetzung für unsere symbolische Simulation untersucht werden. Durch die Bindung von Bedingungen an kontinuierliche Variablen variieren die Länge einer Trajektorie bzw. die Anzahl durchlaufener Zyklen entsprechend unterschiedlicher Parameterbelegungen unvorhersehbar. Der Begriff der symbolischen Simulation kann hier zusammenfassend bezüglich unseres Ansatzes konkretisiert werden.

Begriff 5.3.2
Die **symbolische Simulation** basiert auf der diskret-ereignisorientierten Simulation, bei welcher symbolische Trajektorien als **Zeitwörter** von einem System bestehend aus hybriden Automaten zur Überprüfung zeit- und sicherheitskritischer Bereiche der Synchronisation dieser Automaten akzeptiert werden.

Begriff 5.3.3
Ein **Zeitwort** ist eine endliche bzw. unendliche Folge von Tupeln, in welchen Ereignisse, die als Symbole des Zeitwortes betrachtet werden, mit der symbolischen Zeit ihres Auftretens verbunden sind.

5.3.1 Wörter - Transitionsbasierte Sicht

Die symbolische Simulation aus [RU95, Rie95] geht von der Überprüfung endlicher und unendlicher Zeitwörter auf der Basis einer zustandsbasierten Sichtweise der Definition 2.2.2 aus. Um dem Begriff der Akzeptanz eines Zeitwortes in hybriden Automaten gerecht zu werden, wird in dieser Arbeit eine transitionsbasierte Sicht [Tet08] eingeführt.

Zeitwörter werden als Folge von Transitionen beschrieben, wobei jedes Symbol verbunden mit der symbolischen Zeit seines Auftretens einer Transition in einem hybriden Automaten entspricht. Die symbolische Zeit bestimmt einerseits die Wertebelegung einer Menge von kontinuierlichen Variablen, und andererseits ist die Zeit durch an die Variablen vorgegebene Bedingungen gebunden. Konkret kann ein Zeitwort hier wie folgt definiert werden:

Definition 5.3.1
Ein **Zeitwort** ZW bildet eine endliche Folge von Tupeln
$ZW \quad \langle S_1,\ T_1,\ X_1,\ \omega_1\rangle$,...,$\langle S_n,\ T_n,\ X_n,\ \omega_n\rangle$ mit $i\ =\ 1,..,n$ bzw.
eine unendliche Folge von Tupeln
$ZW \quad \langle S_1,\ T_1,\ X_1,\ \omega_1\rangle$,$\langle S_2,\ T_2,\ X_2,\ \omega_2\rangle$,... mit $i\ =\ 1,2,...$, wobei gilt:

S_i	Menge von Symbolen,
T_i	symbolische Zeit,
X_i	Menge von Variablen,
ω_i	Menge von Abbildungen der Variablen in einen Wertebereich.

Definition 5.3.2
Ein **Zeitwort** heißt **konkret**, wenn alle Abbildungen der Menge ω_i mit $i\ =\ 1,...,n$ bzw. $i\ =\ 1,2,...$ den Variablen konkrete Werte zuweisen.

Definition 5.3.3
Ein **Zeitwort** heißt **symbolisch**, wenn Abbildungen einer Menge ω_i mit $i\ =\ 1,...,n$ bzw. $i\ =\ 1,2,...$ existieren, die den Variablen eine Menge von Werten zuweisen. Der Wertebereich ist dabei unendlich dicht und wird mithilfe von Bedingungen über Intervallen dieser Werte spezifiziert.

Die Menge und Art der Abbildungen ω auf die Variablen richtet sich nach dem zugrundeliegenden Modell. In unserem Fall sind auf der Grundlage hybrider Automaten 4 Abbildungsarten zu berücksichtigen:

1. Invarianten der Lokationen als Relationen (mehrdeutige Abbildungen),
2. Aktivitäten der Lokationen (eindeutige Abbildungen),
3. Bedingungen der Transitionen als Relationen (mehrdeutige Abbildungen),
4. Aktionen der Transitionen (eindeutige Abbildungen).

Da die transitionsbasierte Sicht der Akzeptanz von Zeitwörtern keine explizite Beschreibung von Invarianten und Aktivitäten zulässt, deren Einhalten jedoch für den Eintritt und das Verlassen von Lokationen notwendig sind, wurden die Differentialgleichungen der *Aktivitäten* in *Diffenzengleichungen* und die *Invarianten* entsprechend [BS97] in *unbedingte Transitionsbedingungen* transformiert. Im Unterschied zu Zeitautomaten [AD94,

Alu99] und Zeitpetrinetzen [SY96] bzw. Echtzeitstatecharts [GB03, SASX05] sind unbedingte Transitionsbedingungen hybrider Automaten aus den Invarianten im Zusammenhang mit den dazugehörigen Aktivitäten der Ausgangslokationen herzuleiten. Die Transformation kann bezüglich der folgenden Begriffe mit der Funktion 'uncond_trans' beschrieben werden.

Definition 5.3.4
Ein **Term** T wird induktiv wie folgt beschrieben:

1. Jede Konstante k und jede Variable X sind Terme.
2. Wenn t_1,...,t_n Terme sind und f ein n-stelliges Funktionssymbol, dann ist f(t_1,...,t_n) ein Term.

Definition 5.3.5
Ein **Vergleich** V ist eine 2-stellige Relation über Termen T mit $V \in \{<, \leq, =, \geq, >\}$.

Definition 5.3.6
Jede **Invariante** hat folgende Form:

1. Jeder Vergleich V ist ein Invariante.
2. Wenn a und b Invarianten, dann ist $a \wedge b$ eine Invariante.

Transformation von Invarianten in unbedingte Transitionen mit:

X	kontinuierliche Variable,	f	Funktionssymbol,
T	Term,	$\lhd \in$	$\{=, \leq\}$
$\rhd \in$	$\{=, \geq\}$,	ϵ	kleinst möglicher Wert größer Null

define function uncond_trans($Invariante$, $Aktivitäten$)
  if $Invariante$ true **then** $Invariante$
  else
    let $Invariante$ $Vergleich \wedge InvRest$ **in**
      transform($Vergleich$, $Aktivitäten$) $\wedge$ uncond_trans($InvRest$, $Aktivitäten$);

define function transform($Vergleich$, $Aktivitäten$)
  let X cont_var($Vergleich$) **in**
    if derivation(X, $Aktivitäten$) > 0 **then**
      if form_eq(Vergleich, f(X) $\lhd$ T) **then**
        f(X) $\lhd$ $T + \epsilon$;
      else if form_eq(Vergleich, f(X) $<$ T) **then**
        f(X) $\leq$ T;
      else $Vergleich$;
    else if derivation(X, $Aktivitäten$) < 0 **then**
      if form_eq(Vergleich, f(X) $\rhd$ T) **then**

f(X) $\rhd$ $T - \epsilon$;
else if form_eq(Vergleich, f(X) $>$ T) **then**
f(X) $\geq$ T;
else $Vergleich$;
else $Vergleich$;

Als Voraussetzung ist angenommen, dass mit jedem Vergleich der Wertebereich genau einer kontinuierlichen Variablen 'X' einschränkt wird. Zur Transformation von 'Vergleich' ist der Anstieg der zu 'X' gehörigen Aktivität ausschlaggebend. Im Fall eines positiven Anstiegs ist mit 'form_eq' zu überprüfen, ob in dem Vergleich der Wert der Variablen 'X' gleich bzw. kleiner gleich einem mit einem Term 'T' berechneten Wert zum Verbleib innerhalb der betrachteten Lokation ist. Das Verlassen der Lokation ist während des Intervalls 'f(X) $\lhd$ A' möglich, aber auch noch bei Überschreiten der Intervallgrenze 'T' um ϵ. Deshalb ist zur Entwicklung der unbedingten Transitionsbedingung der 'Vergleich' von 'f(X)$\lhd$T' nach 'f(X)$\lhd$T+ϵ' zu ersetzen. Ist der Wert der Variablen 'X' kleiner dem Wert des Termes 'T', so ist das Intervall 'f(X) < T' durch das Intervall 'f(X)$\leq$ T' zu ersetzen. Ist der Anstieg der Aktivität zur Berechnung der Werte für 'X' kleiner als Null, so werden die unteren Grenzen im Vergleich betrachtet. Ist der Anstieg gleich Null, so braucht keine separate Behandlung zu erfolgen, da die Invarianten bei Eintritt in die Lokation bereits überprüft wurden und sich die Werte der kontinuierlichen Variablen 'X' nicht verändert haben.
Im Gegensatz zu [Sta00] wird in unserem Ansatz gefordert, dass die Invariante der Lokation, welche erreicht wird, zum Zeitpunkt des Eintritts erfüllt ist. Ein zeitgleiches Verlassen der Lokation bei deren Eintritt ist möglich.
Bezüglich hybrider Automaten lässt sich die Menge der Abbildungen ω mit folgender Grammatik formalisieren:

```
<ω> ::= ⟨ <VorTrans>, <NachTrans> ⟩
<VorTrans> ::= release( <Aktivitäten>, <TransBeding> )
<TransBeding> ::= <UnbedingtTrans> ∧ <BedingtTrans>
<NachTrans> ::= conclude( <Aktionen>, <EintrittInv> )
```

Die Menge der Abbildungen ω setzt sich aus einer Menge von Abbildungen zum Auslösen einer Transition in 'VorTrans' und einer Menge von Abbildungen zum Abschließen einer Transition in 'NachTrans' zusammen. Um eine Transition auszulösen, muss die Wertebelegung der Variablen zum Zeitpunkt T, welcher gleichzeitig den Zeitpunkt des Verlassens der Vorgängerlokation darstellt, durch die Differenzengleichungen der Aktivitäten 'Aktivitäten' der Vorgängerlokation berechnet werden. Für diese berechneten Werte muss die Bedingung 'TransBeding' gelten. Die Bedingung 'TransBeding' bildet dabei die Konjunktion aus einer unbedingten Transitionsbedingung 'UnbedingtTrans' und der explizit für die im Automaten gegebenen Transitionsbedingung 'BedingtTrans'. Die Transition muss ausgeführt werden, wenn die Variablenwerte den sofort auslösenden Bedingungen aus 'UnbedingtTrans' entsprechen und kann ausgeführt werden, wenn die Variablenwerte

im Bereich der Bedingung 'BedingtTrans' liegen. Die Transition kann erfolgreich abgeschlossen werden, wenn alle Aktionen aus 'NachTrans' ohne Widersprüche vollständig ausgeführt sind und die Invariante 'EintrittInv' der Nachfolgelokation gilt.
Anhand des Synchronisationsproduktes aus Abbildung 4.14 für die Darstellung des Prozesses eines Studienbüros in Verbindung mit dem in einer Datenbank zu verwaltenden Prüfungsablauf ist im folgenden Beispiel ein durch den SHA 'Pruefung_Bearbeitung' spezifiziertes Wort aufgelistet.

Beispiel 5.3.1
Die Variable T ist für jede Transition mit einer fortlaufenden Nummer indiziert. Für die Variablen der Uhren S, A und B gilt:

$'S, 'A, 'B$ *Werte der Variablen bei erfolgreichem Abschluss der vorherigen Transition*
S', A', B' *Werte der Variablen bei erfolgreichem Abschluss der betrachteten Transition.*

Diese Unterscheidung wurde hier vorgenommen, um syntaktisch deutlich hervorzuheben, wann welcher Wert einer Variablen verwendet wird. Auch für die Ausführung der symbolischen Simulation in CLP besteht die Notwendigkeit der Unterscheidung, wofür Gründe im Abschnitt 5.3.3 angegeben werden.

$\langle$ {**Beginn**↓}, $T[0]$ {S, A, B},
$\langle$*release({}, 'true'),*
conclude({S': 0, A': 0, B': 0},$0 \leq S' \wedge 0 \leq A' \wedge 0 \leq B' < 22$)$\rangle\rangle$,
$\langle$ {**DB_Vor_b**↓,**Info_Vor_b**↑}, $T[1]$, {S, A, B},
$\langle$*release({S: $'S$+(T[1]-T[0]), A: $'A$+(T[1]-T[0]), B: $'B$+(T[1]-T[0])},*
($0 \leq S \wedge 0 \leq A \wedge 0 \leq B \leq 22$) $\wedge$ ($A < 20 \wedge B < 22$)),
conclude({S': 0},$0 \leq S' \wedge 0 \leq A' < 24 \wedge 0 \leq B' < 24$)$\rangle\rangle$,
$\langle$ {**DB_Pruef_b**↓,**Info_Pruef_b**↑}, $T[2]$, {S, A, B},
$\langle$*release({S: $'S$+(T[2]-T[1]), A: $'A$+(T[2]-T[1]), B: $'B$+(T[2]-T[1])},*
($0 \leq S \wedge 0 \leq A \leq 24 \wedge 0 \leq B \leq 24$) $\wedge$ ($22 \leq A < 24 \wedge 22 \leq B < 24 \wedge S \leq 2$),
conclude({A': 0, B': 0},$0 \leq S' \wedge 0 \leq A' \wedge 0 \leq B'$)$\rangle\rangle$

Die spezifizierte Folge von Symbolen in Zusammenhang mit einer Zeitvariable sowie Variablen, Aktivitäten, unbedingten und bedingten Transitionsbedingungen, Aktionen und Eingangsbedingungen entsprechend der Transitionen eines hybriden Automaten im obigen Beispiel zeigt die Bildung eines Wortes auf der syntaktischen Ebene. Um jedoch explizit Aussagen über:

1. Bedingungen an die Zeit und damit
   - die Akzeptanz von Wörtern bzw.
   - die Entscheidbarkeitsfrage in hybriden Automaten und
   - die Erreichbarkeit von Lokationen,

2. Inkonsistenzen von Bedingungen an Variablen,
3. Zusammenhänge zwischen Wertebelegungen von Variablen,
4. Interpretationen:
    - zur Verbesserung und weiteren Entwicklung eines Modells sowie
    - zu praktischen Problemen in der Anwendung

treffen zu können, muss eine Semantik zur Auswertung der Wörter und Beantwortung der Fragestellungen geschaffen werden. Dazu wird in unserem Ansatz die symbolische Simulation mithilfe von CLP (Constraint-Logische Programmierung) aus der transitionsbasierten Sicht auf hybriden Automaten entwickelt. Im Kapitel 7 wird später auf die Formulierung von Anfragen, Interpretationen von Ergebnissen der symbolischen Simulation und deren Anwendung eingegangen. In den nächsten Abschnitten erfolgt eine Beschreibung des constraint-logischen Paradigmas und daran anschließend die Umsetzung des Algorithmus der symbolischen Simulation und ein Beispiel.

5.3.2 Beschreibung von CLP

Die constraint-logische Programmierung [JM94, FA97, HW07] bildet die Grundlage der Beschreibung hybrider Systeme und zugehöriger zeit- und sicherheitskritischer Eigenschaften in Form von Zeitwörtern. Der Ausführungsmechanismus von CLP dient der symbolischen Simulation und somit der semantischen Auswertung von Beziehungen und Bedingungen der Variablen von Zeitwörtern. Durch die Ausführung in CLP kann dabei über Anfragen ermittelt werden, ob:

1. eine als Zeitwort formulierte Eigenschaft von dem hybriden System akzeptiert wird bzw.
2. welche Zeitwörter unter bestimmten Umgebungsbedingungen teilweise bzw. vollständig zur Sprache des hybriden Systems gehören.

Wie die Umgebungsbedingungen aussehen und welche Form die Anfragen besitzen, kann im Abschnitt 7.2 nachgelesen werden. In der constraint-logischen Programmierung werden zwei Paradigmen vereinigt:

- Lösen von Constraints und
- Logische Programmierung.

An dieser Stelle soll kurz auf den Vorteil der Verbindung dieser beiden Paradigmen eingegangen werden. Weiterhin sind Constraintsysteme und das diesen Systemen zugrundeliegende Kalkül der abstrakten Syntax und operationalen Semantik entsprechend [FA97] erklärt.

Von der Logischen zur Constraint-logischen Programmierung

Für Constraints wird eine Definition wie folgt angegeben:

Definition 5.3.7
Constraints sind Bedingungen, die in der Form von Constraintprädikaten $p(t_1, ..., t_n)$ angegeben werden, wobei gilt:

1. $p \in \mathcal{C}, \mathcal{C}$ *Menge der Symbole* der Constraintprädikate, die bezüglich einer vordefinierten Struktur interpretiert werden,

2. $t_i,\ i \in \{1, ..., n\}$ sind *Terme*.

Constraints können auf diese Art und Weise unbekannte Bereiche beschreiben. In constraint-logischen Sprachen besitzen die Constraints folgende Merkmale [JM94]:

- Constraints werden sowohl zur Spezifikation von Anfragen als auch von deren Antworten genutzt.

- Während der Ausführung werden neue Variablen und Constraints geschaffen.

- Die Sammlung von Constraints wird in jedem Zustand im Ganzen auf eine Lösung getestet, bevor die Ausführung der Regeln fortgesetzt wird.

Um zu verdeutlichen, welchen Vorteil die constraint-logische Programmierung gegenüber der logischen Programmierung besitzt, ist hier ein Beispiel aus [JM94] aufgeführt, welches den Unterschied dieser Programmierungstechniken aufzeigt. Es besteht die Frage, ob die logischen Programmiersprachen die Leistungsfähigkeit der constraint-logischen Programmiersprachen erreichen können.

Beispiel 5.3.2
Gegeben sind die folgenden Regeln:

```
add(0, N, N).
add(s(N), M, s(K)) :- add(N, M, K).
```

Die natürlichen Zahlen n werden bei diesen Regeln in der Form s(s(...(0)...)) *mit n - maligem Auftreten von* s *repräsentiert. Diese Regelstruktur kann als Addition interpretiert werden.*
Wird an das System die Anfrage gestellt:

```
add(N, M, K), add(N, M, s(K)). ,
```

so werden die Regeln unendlich lange verarbeitet, ohne zu dem Ergebnis der Unlösbarkeit dieser Anfrage zu gelangen. Das Problem bei dieser Anfrage besteht darin, dass die Regeln in keiner Weise mit einer Repräsentation eines Constraints in Verbindung stehen. D.h. ein Atom wie add(N, M, K) *in der Regel* add(s(N), M, s(K)) :- add(N, M, K) *repräsentiert nicht den Fakt* N+M K.
Eine Teillösung kann in logischen Programmiersprachen durch die Nutzung von Verzögerungsmechanismen erreicht werden. Der Aufruf von Prädikaten wird dabei solange zurückgestellt, bis dessen Argumente ausreichend instantiiert sind. Die Anfrage:

N s(s(...(0)...)), add(N, M, K), add(N, M, s(K)).

schlägt dementsprechend fehl. Doch die ursprüngliche Anfrage add(N, M, K), add(N, M, s(K)). *wird unendlich lange verzögert. Eine vollständige Lösung des Problems kann durch die zusätzliche Verwendung zweier Argumente in den Regeln gefunden werden. Hierbei stellt das erste Argument die Eingangsrepräsentation des Prädikates dar und das zweite Argument die Ausgaberepräsentation. Die Einführung derartiger Argumente hat zur Folge, dass zu jeder Zeit ein Constraint mitbehandelt werden muss. Aus der Repräsentation einer ganzen Menge von berechneten Constraints wird eine neue Repräsentation konstruiert.*

Die vollständige Lösung in der logischen Programmierung bildet gerade den Ansatz, welche die constraint-logische Programmierung mit effizienteren Methoden, wie z.B. der Simplexmethode, verfolgt. Solche Methoden werden in speziell dafür vorgesehenen Constraintlösern genutzt.

Constraintsysteme

Zur formal logischen Beschreibung von constraint-logischen Programmiersprachen wurden verschiedene Modelle entwickelt. Ein Modell ist das CLP - Schema, welches in [JL87] erstellt wurde. Es besitzt nur über einem einzigen Werte- bzw. Constraintbereich Gültigkeit. Ein allgemeineres Schema, das Höhefeld-Smolka-Schema, wurde in [HS88] vorgestellt, welches Constraints über mehrere Wertebereiche zulässt.
Ein Constraintsystem auf der Grundlage des Höhefeld-Smolka-Schemas wird wie folgt definiert:

Definition 5.3.8
Ein **Constraintsystem** ist ein Tripel (Σ, CT, C). Dabei sind:

- Σ eine *Signatur*,
- CT eine *Constrainttheorie*, welche eine nichtleere Formelmenge über Σ beschreibt und konsistent ist,
- C die Menge der erlaubten Constraints, welche Formeln über Σ derart enthält, dass

- $true \in C$ und $false \in C$
- C ist abgeschlossen unter Konjunktion und Variablenumbenennung.

Die Constrainttheorie CT schränkt die möglichen Interpretationen für Constraintsymbole über die Definition von Axiomen ein. Die Menge C ermöglicht es, Constraints effizient als Ziele in Klauseln zu nutzen.

Beispiel 5.3.3
Nach [FA97] sieht das Herbrand System, welches den Fakten und Klauseln einer constraint-logischen Sprache zugrundeliegen kann, wie folgt aus:

> Signatur Σ*:*
> *unendlich viele Funktionssymbole F mit mindestens einer Konstante k, einem zweistelligen Constraintsymbol $\doteq$, den nullstelligen Constraintsymbolen* true *und* false*,*
>
> Constrainttheorie CT*:*
> *Clarksche Gleichheitstheorie [FA97], die bezüglich des Operationssymbols $\doteq$ mit Reflexivität, Symmetrie, Transitivität, Verträglichkeit, Zerlegung, Widerspruch und Azyklizität erklärt ist,*
>
> Menge erlaubter Constraints C*:*
> $C :: \ true \mid false \mid s \doteq t \mid C \wedge C$*, wobei s und t beliebige Terme über der Signatur Σ sind.*

Alle aus Variablen, Atom- und Prädikatssymbolen gebildeten Terme werden in die Domäne der Bäume abgebildet. Ein Constraintlöser implementiert die Clarksche Gleichheitstheorie bezüglich des Herbrand Systems folgendermaßen: Wenn zwei Terme s und t unifizierbar sind und somit das Gleichheitsconstraint $s \doteq t$ erfüllbar ist, dann wird $s \doteq t$ zu der Normalform $X_1 \doteq t_1 \wedge ... \wedge X_n \doteq t_n$ mit den paarweise verschiedenen Variablen X_1, ... ,X_n, die nicht in den Termen t_1, ... ,t_n vorkommen, vereinfacht. Sonst wird $s \doteq t$ zu $false$ vereinfacht.

Formale Beschreibung

In diesem Abschnitt wird anhand eines Kalküls aus [FA97] die Syntax der constraint-logischen Programmierung und die operationale Semantik, nach welcher die Programme der constraint-logischen Programmiersprachen ausgeführt werden, beschrieben.

Abstrakte Syntax

Die Syntax der constraint-logischen Programmierung ist auf der Grundlage der Syntax der logischen Programmierung durch die Erweiterung um zusätzliche Constraints entstanden. Die **Syntax** besteht aus folgenden Elementen, wobei die hervorgehobenen Elemente die Veränderung gegenüber der Syntax der logischen Programmierung darstellen:

Atome:	A	::	$p(t_1, ..., t_n),\ n \geq 0$
Constraints:	C, D	::	$c(t_1, ..., t_n) \mid C \wedge D,\ n \geq 0$
Ziele:	G, H	::	$\top \mid \bot \mid A \mid C \mid G \wedge H$
Klauseln:	K	::	$A \leftarrow G$
Programme:	P	::	$\{K_1, ..., K_m\},\ m \geq 0.$

Hierbei sind $p(t_1, ..., t_n)$ Atome und $c(t_1, ..., t_n)$ Constraintatome, wobei p ein Prädikatssymbol, c ein Constraintsymbol und $t_1, ..., t_n$ Terme sind. $\top$ (Top) und $\bot$ (Bottom) bilden die nullstellig logischen Verknüpfungen.

Operationale Semantik

Die Ausführung eines constraint-logischen Programmes P wird als eine Folge von Zuständen betrachtet. Die Übergänge werden als Reduktionsschritte behandelt, die unter bestimmten Bedingungen vorgenommen werden. Solche Übergänge, die unter den Bedingungen $C_i, i \in \{1, ..., n\}$ von einem Zustand S in einen Nachfolgezustand S' erfolgen, können mit folgenden Ableitungsregeln beschrieben werden:

$$\frac{\begin{array}{c} C_1 \\ \cdots \\ C_n \end{array}}{S \mapsto S'} \ .$$

Ein *Zustand* ist ein Paar $\langle G,\ C \rangle$, wobei G ein Ziel und C ein Constraint darstellt. G wird Zielspeicher und C Constraintspeicher genannt. $\top$ stellt in diesem Fall den leeren Zielspeicher dar. Die folgenden Zustände liegen unter den beschriebenen Bedingungen vor:

Anfangszustand	$\langle G,\ true \rangle$
erfolgreicher Endzustand	$\langle \top,\ C \rangle$, C ungleich false
erfolgloser Endzustand	$\langle G,\ false \rangle$

Die Ableitungen, welche zu diesen Zuständen führen können, sehen wie folgt aus:

Entfalten:

$$\frac{\begin{array}{c} (B \leftarrow H)\ aus\ P \\ CT \models \exists((B \doteq A) \wedge C) \end{array}}{\langle A \wedge G,\ C \rangle \mapsto_{Entfalten} \langle H \wedge G,\ (B \doteq A) \wedge C \rangle}$$

Scheitern:

$$\frac{\begin{array}{c} Es\ gibt\ keine\ Klausel\ (B \leftarrow H)\ aus\ P\ mit \\ CT \models \exists((B \doteq A) \wedge C) \end{array}}{\langle A \wedge G,\ C \rangle \mapsto_{Scheitern} \langle \bot,\ false \rangle}$$

Vereinfachen:

$$\frac{CT \models (C \wedge D_1) \leftrightarrow D_2}{\langle C \wedge G,\ D_1\rangle \mapsto_{Vereinfachen} \langle G,\ D_2\rangle}$$

Wenn zu einem Atom A einer Anfrage eine Klausel mit dem Klauselkopf B derart existiert, dass die Gleichheit von A und B zusammenen mit dem vorliegenden Constraintspeicher C konsistent ist, so kann das Atom A durch den Klauselrumpf H ersetzt und der Gleichheitsconstraint zum Constraintspeicher C hinzugefügt werden. Diese Reduktion wird als *Entfaltung* bezeichnet. Liegen die genannten Voraussetzungen nicht vor, so führt dies zum *Scheitern* der Anfrage.
Besteht die Anfrage aus einem Constraint C, so wird C beim *Vereinfachen* in Konjunktion mit dem vorliegenden Constraintspeicher D_1 zu D_2 vereinfacht. Das Ziel C wird dabei aus dem Zielspeicher entfernt und der Constraintspeicher enthält den vereinfachten Constraint. In der Regel erfolgt eine Überprüfung der Konsistenz der Konjunktion von C und D_1, wobei inkonsistente Constraints zu 'false' vereinfacht werden. Diese Vereinfachung führt zu einem erfolglosen Endzustand wie das Scheitern.

5.3.3 Symbolische Simulation in CLP

Nachdem die Syntax anhand verwendeter Fakten und Klauseln sowie deren denotationale Semantik beschrieben wurde, wird in einem weiteren Abschnitt die operationale Semantik anhand der Regeln des Entfaltens, Vereinfachen und Scheiterns aus CLP dargestellt.

Syntax und denotationale Semantik

Entsprechend der Definition 5.3.8 für Constraintsysteme wird unsere symbolische Simulation hybrider Systeme über 4 Systemen von Constraints ausgeführt:

1. Herbrand Constraints laut Beispiel 5.3.3,
2. Listen Constraints - Elementsymbole E und Listensymbole L mit Operationssymbolen wie 'empty', 'eq', 'in', 'concat' sowie Axiomen zur Definition von Eigenschaften und erlaubten Constraints wie z.B. empty(s),...,concat(s, t, u) mit s, t, $u \in L$, wobei alle aus E und L gebildeten Terme in die Domäne der Bäume abgebildet werden,
3. Numerische Constraints - Elementsymbole R mit Operationssymbolen für alle Operationen über der Domäne der reellwertigen Zahlen,
4. Boolsche Constraints - Elementsymbole B mit Operationssymbolen für alle Operationen über der Domäne der boolschen Werte.

Die symbolische Simulation ist von 5 Eingangsparametern abhängig:

1. der *Eigenschaft*, die als 'Zeitwort' formuliert wurde,

2. dem *hybriden System* 'HybSys', dessen Regeln vorher analysiert und in den Speicher geladen werden,

3. der *Lokation* 'Aktiv_Lok', die zu Beginn der Ausführung aktiv ist,

4. der *Zeit* 'Tstart', zu welcher die Ausführung beginnt,

5. der *Variablenbelegung* 'VARstart', die zu Beginn der Ausführung vorliegt.

Im Allgemeinen stimmen 'Tstart' und 'VARstart' mit der Zeit und der Variablenbelegung der ersten Symbolmenge des Zeitwortes überein. Auf diese Art und Weise besteht die Möglichkeit, explizit für mehrere Durchläufe Standardwerte festzulegen. Im Abschnitt 7.2 des Kapitels 'Weitere Arbeiten' wird nochmal einmal ausführlicher darauf eingegangen.
Der Begriff der symbolischen Simulation in CLP lässt sich wie folgt fassen.

Begriff 5.3.4
Die **symbolische Simulation in CLP** ist durch die Abarbeitung einer Anfrage
'query(*Eigenschaft, Hybrides System, Startlokation, Startzeit, StartVarBelegung*)'
als Ziel G nach:
der Erfüllbarkeit einer Eigenschaft in Form eines Zeitwortes
an ein hybrides System in Form einer Menge von Regeln in CLP
vom Anfangszustand $\langle G, true \rangle$ festgelegt. Während der Ausführung werden an die Zeiten und Variablen der als Zeitwort vorliegenden Eigenschaft Bedingungen des Nutzers als auch des hybriden Systems gebunden, die als Constraintsystem im Constraintspeicher C durch den Constraintlöser vereinfacht, gelöst bzw. zu Widersprüchen geführt werden.

Begriff 5.3.5
Ein **akzeptiertes Zeitwort** ist ein Wort, welches bei der Ausführung der symbolischen Simulation rekursiv bis zum leeren Wort traversiert wird und dessen Abarbeitung zu einer Endlokation des hybriden Systems führt.

Das Programm P setzt sich aus Regeln zur Beschreibung des hybriden Systems 'HybSys' und des Simulationsvorganges zusammen. Das hybride System wird dabei über folgende Fakten realisiert:

Synchronisationsverbindungen:

init_connect(*Anfangsbedingung, Anfangsaktion***).**
connect(*HHAsend,* receive(*RSymbole*)*, Bedingung,*
send(*SSymbole*)*, Aktion*, *HHAreceive***).**

Lokationen und Transitionen:

endlocations(*Lokationsnamen***).**
location(*Name, Invariante, Aktivitäten***).**
transition(*VorLokation,* receive(*RSymbole*)*, TransBedingung,*
send(*SSymbole*)*, TransAktion, NachLokation***).**

Hierarchisch hybride Automaten liegen bereits in flachen Automaten mit eindeutigen Namen der Variablen, Symbole und Lokationen vor. Synchronisierend hybride Automaten auf synchronisierend hybriden Automaten wurden in einen synchronisierend hybriden Automaten überführt, so dass als Ergebnis ein synchronisierend hybrider Automat über einfachen hierarchisch hybriden Automaten für die symbolische Simulation zugrunde gelegt wird. Ein Fakt für eine Anfangslokation wird nicht benötigt, da Anfangslokationen standardmäßig mit der Invariante 'true' und der leeren Menge an Aktivitäten ausgestattet sind.
Zur Darstellung des Prozesses der symbolischen Simulation werden die Bezeichnungen 'aktive Lokation' und 'aktive Transitionen' verwendet, die hier begrifflich gefasst werden.

Begriff 5.3.6
Eine **aktive Lokation** bezüglich eines gegebenen Zeitintervalls TI ist diejenige Lokation, die während TI betrachtet wird.

Begriff 5.3.7
Aktive Transitionen bezüglich eines gegebenen Zeitpunktes TP sind all diejenigen Transitionen, die zum Zeitpunkt TP von der aktiven Lokation ausgehen und deren Symbole in der Menge der Symbole des zum Zeitpunkt TP betrachteten Tupels einer Eigenschaft enthalten sind.

Der Simulationsvorgang wird durch Prädikate folgender Struktur beschrieben:

symb_sim(*Zeitwort, Aktiv_Lok, Tvor, VARvor***)**.

Ist das Zeitwort leer, so wird überprüft, ob die gerade aktive Lokation eine Endlokation des Systems darstellt. Das Prädikat sieht in diesem Fall wie folgt aus:

symb_sim(*[], Aktiv_Lok, Tvor, VARvor***)**
:-
in_end(*Aktiv_Lok,Endlokationen***)**.

Besitzt das Zeitwort noch abzuarbeitende Elemente in Form von Tupeln ⟨*Symbole, T, [VAR, VARnach], Bedingung*⟩, so gilt folgende Klausel:

symb_sim(*[⟨Symbole, T, [VAR, VARnach], Bedingung⟩|Restwort],*
*Aktiv_Lok, Tvor, VARvor***)**
:-
combi_active_trans(*⟨Symbole, T, [VAR, VARnach], Bedingung⟩,*
*Aktiv_Lok, Nachf_Lok, Tvor, VARvor***),**
symb_sim(*Restwort, Nachf_Lok, T, VARnach***).**

In der Implementation des Algorithmus zur symbolischen Simulation in CLP ist eine Unterscheidung von Variablenbelegungen bezüglich der Ausführung von Transitionen unter folgenden Gesichtspunkten vorgenommen worden:

1. Variablen in CLP stehen im mathematischen Sinn für einen Wert und nicht wie im imperativen Paradigma je nach Position für einen Wert selbst bzw. als Platzhalter für einen Wert.

2. Werte von Variablen in CLP sind mittels Constraints über Gleichungen und Ungleichungen, jedoch nicht durch Zuweisungen bestimmbar.

3. Aus semantischer Sicht ist es ausreichend, eine Menge 'VAR' von Variablenbelegungen und deren Nachfolgebelegungen 'VARnach' anzugeben. Zur besseren Verständlichkeit des Ablaufes wurde auf syntaktischer Ebene eine Menge 'VARvor' bezüglich jeder Transition eingeführt, welche die Variablenbelegung bei Abschluss der vorhergehenden Transition beinhaltet und eine Kopie von 'VARnach' dieser vorhergehenden Transition bildet.

In Bezug auf Beispiel 5.3.1 gehören alle Variablen mit einem Hochkomma vor dem Bezeichner zur Liste 'VARvor' und alle Variablen mit dem Hochkomma nach dem Bezeichner zur Liste 'VARnach'. Diese geänderten Bezeichnungen wurden zur Beschreibung der Regeln aus Gründen eines besseren Überblicks verwendet. Für das gerade betrachtete Tupel '⟨Symbole, T, [VAR, VARnach], Bedingung⟩' des Zeitwortes werden durch das Prädikat 'combi_active_trans' sämtliche Constraints in Bezug auf die Zeiten 'T' und 'Tvor' bzw. Variablenlisten 'VARvor', 'VAR' und 'VARnach' gesammelt, die zu den aktiven Transitionen bezüglich der Zeit 'T' gehören. Die Konjunktion der gesammelten Constraints wird dem Constraint des Tupels 'Bedingung' gleichgesetzt. Weiterhin erfolgt durch 'combi_active_trans' die Berechnung der Nachfolgelokation 'Nachf_Lok', die der Simulation des verbleibenden Wortes 'Restwort' als neue aktive Lokation übergeben wird. Die Zeit und die Variablenbelegung, auf die bei der weiteren Bearbeitung des Restwortes Bezug genommen werden muss, ist die Zeit 'T' und die Variablenbelegung 'VARnach' des gerade erfolgreich abgearbeiteten Tupels.
Die vollständige Klausel für 'combi_active_trans' wird durch Zielatome der Prädikatssymbole 'synchron', 'release' und 'conclude' bestimmt:

combi_active_trans(⟨*Symbole, T, [VAR, VARnach], Bedingung*⟩,
*Aktiv_Lok, Nachf_Lok, Tvor, VARvor***)**
:-
synchron(*Symbole, VAR, VARnach, Aktiv_Lok, SynBeding, SynAktion***),**
release(*Symbole, T, Tvor, VAR, VARvor, Aktiv_Lok, Nachf_Lok,*
*Aktivitäten, UnbedingtTrans, BedingtTrans***),**
conclude(*Symbole, VAR, VARnach, Nachf_Lok, Aktionen, EintrittInv***).**

Mit dem Prädikat 'synchron' wird auf der Grundlage der transitiven Hülle des Abschnittes 4.4 die Menge der Symbole ermittelt, die aufgrund der Transitivität von Synchronisationsbeziehungen an der Gesamttransition beteiligt sein müssen. Stimmt die ermittelte Symbolmenge mit der Menge der gegebenen Symbole überein, so werden sämtliche Bedingungen der Synchronisationsverbindungen, die in den Fakten 'connect' des hybriden Systems 'HybSys' spezifiziert sind, konjunktiv verknüpft an die Variable 'SynBeding' gebunden und sämtliche Aktionen der Synchronisationsverbindungen an die Variable 'SynAktion'.
Das Prädikat 'release' ermittelt die nachfolgende Lokation 'Nachf_Lok' und bindet alle Aktivitäten der Lokation 'Aktiv_Lok' an die Variable 'Aktivitäten', alle aus den Invarianten abgeleiteten unbedingten Transitionsbedingungen an die Variable 'UnbedingtTrans' und alle Transitionsbedingungen der aktiven Transitionen an die Variable 'BedingtTrans'.
Durch das Prädikat 'conclude' werden sämtliche Aktionen der aktiven Transitionen gesammelt und konjunktiv verknüpft an die Variable 'Aktionen' gebunden sowie alle Invarianten der nachfolgenden Lokation an die Variable 'EintrittInv'.
Die entstandenen Constraints in 'SynBeding', 'SynAktion', 'Aktivitäten', 'UnbedingtTrans', 'BedingtTrans', 'Aktionen' und 'EintrittInv' bilden zusammen die Bedingungen über der Zeit 'T' und den Variablen 'VARvor', 'VAR' und 'VARnach', unter denen die Symbole des betrachteten Tupels akzeptiert werden können. Mit der Variable 'Bedingung' des Tupels können durch einen Nutzer zusätzliche Einschränkungen über der Zeit 'T' und den Variablen 'VAR' unabhängig vom hybriden System beschrieben werden. Für die Zeit 'T' und die Variablenbelegungen der Listen 'VAR' und 'VARvor' sind hiermit Constraints gegeben, die in Verbindung mit den Constaints der Zeiten und Variablenbelegungen aller anderen Tupel des Zeitwortes mittels Konsistenztests und Vereinfachungen eines Constraintlösers Aussagen über die Akzeptanz der Eigenschaft als Zeitwort in dem hybriden System 'HybSys' zulassen. Im Vergleich zur beschriebenen Grammatik der Abbildungen ω der Tupel eines Zeitwortes im Abschnitt 5.3.1 sind hier die Verbindungsbedingungen und -aktionen 'SynBeding' und 'SynAktion' hinzugekommen, da der synchronisierend hybride Automat für die symbolische Simulation in seiner ursprünglichen Beschreibung genutzt wird. Im Abschnitt F.1 des Anhangs F sind detaillierte Klauseln für die Prädikate 'release' und 'conclude' angegeben.

Operationale Semantik

Mit der operationalen Semantikbeschreibung wird der Verlauf der symbolischen Simu-

lation anhand der Regeln 'Entfalten', 'Scheitern' und 'Vereinfachen' aus [FA97] noch einmal formal dargestellt und in einem weiteren Abschnitt durch ein Beispiel belegt. In der Abbildung 5.1 ist die Abarbeitung einer Anfrage zur symbolischen Simulation unter folgenden Voraussetzungen dargestellt:

- Aus Gründen der Übersichtlichkeit sind nur Pfade aufgeführt, in denen Klauseln erfolgreich abgearbeitet werden, wobei zwischen den Bedingungen *SynBeding*, *SynAktion*, *Aktivitäten*, *UnbedingtTrans*, *BedingtTrans*, *Aktionen*, *EintrittsInv* und *Beding* keine Widersprüche auftreten. Das Scheitern aufgrund von Widersprüchen wird durch die später beschriebene Abbildung 5.2 genauer dargestellt.

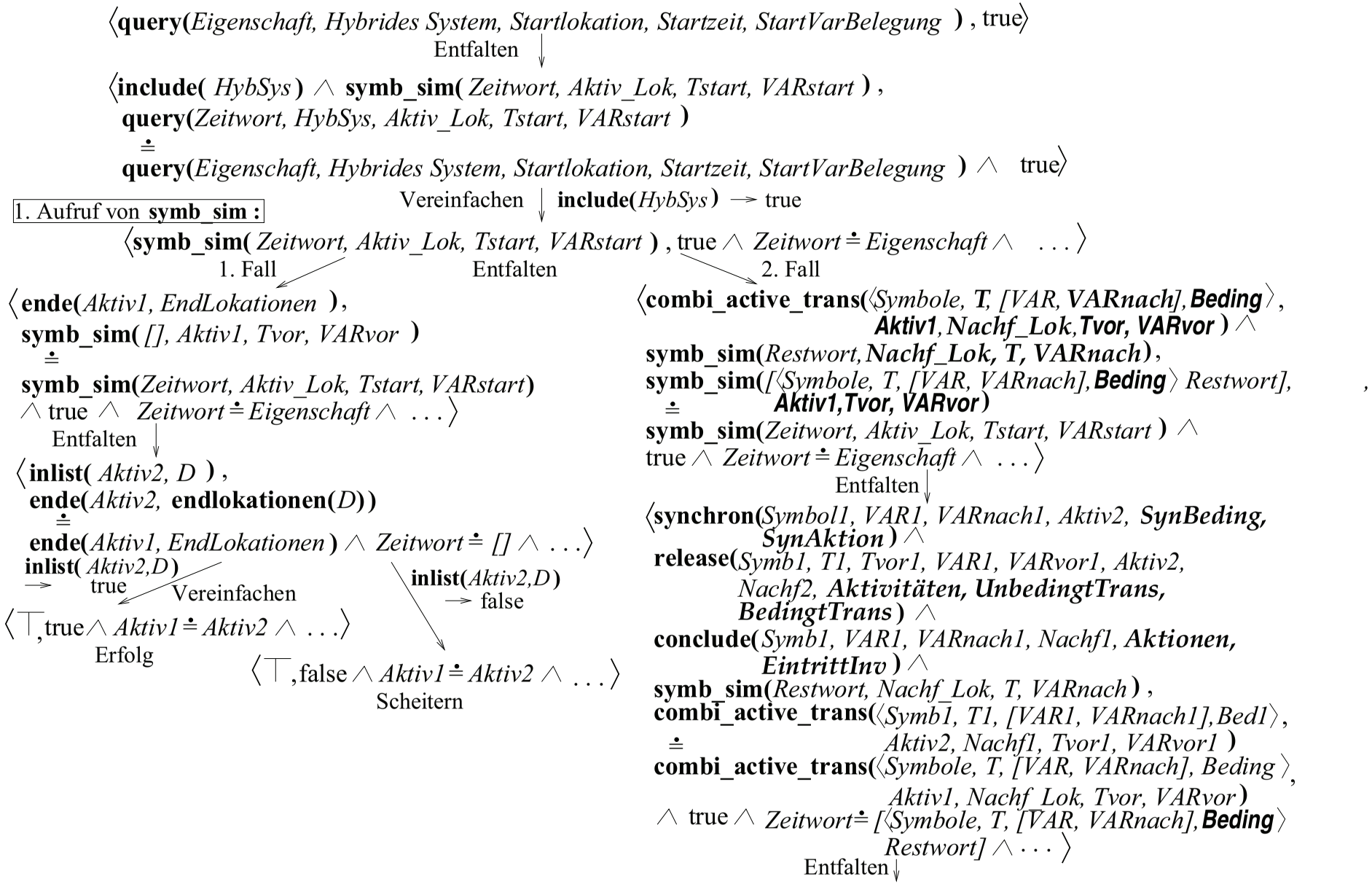
- Das Constraintatom 'include' wird zu 'true' ausgewertet, wenn die Fakten und Klauseln einer mit dem Namen *HybSys* bezeichneten Datei erfolgreich in den Speicher geladen wurden.

- Das Constraintatom 'inlist' wird zu 'true' ausgewertet, wenn ein Element *Aktiv_Lok* in einer Liste *D* enthalten ist.

- Werte von *Aktiv1*, *Tvor* und *VARvor* und an diese Variablen gebundene Bedingungen werden zur Berechnung der Werte und Bedingungen der Variablen *T*, *VAR* und *VARnach* von Zielklauseln ererbt. Werte und Bedingungen von *Nachf_Lok*, *T* und *VARnach* werden in den Zielklauseln synthetisiert und zur Bearbeitung des verbleibenden Wortes *Restwort* kopiert.

- Auf der einen Seite liegen die durch das hybride System gegebenen Bedingungen *SynBeding*, *SynAktion*, *Aktivitäten*, *UnbedingtTrans*, *BedingtTrans*, *Aktionen* und *EintrittsInv* zur Berechnung vor. Auf der anderen Seite hat der Nutzer die Möglichkeit, über *Beding* zusätzliche Bedingungen an die Zeiten und Variablen des Wortes zu stellen.

Als Ziel wird eine Anfrage 'query' formuliert, die die notwendigen Eingangsparameter für die symbolische Simulation enthält. Der Constraintspeicher ist zu Beginn mit dem Wert 'true' belegt. Zu dem Zielatom kann im Programm eine Klausel mit dem Klauselkopf 'query(*Zeitwort, HybSys, Aktiv_Lok, Tstart, VARstart*)' gefunden werden. Durch den Klauselrumpf wird die Anfrage in das Laden der Fakten und Klauseln von *HybSys* durch 'include' und in die symbolische Simulation mit 'symb_sim(*Zeitwort, Aktiv_Lok, Tstart, VARstart*)' entfaltet. Nachdem das hybride System im Speicher vorliegt, wird das Zielatom 'symb_sim' abgearbeitet. Entweder entspricht das Zeitwort des Zielatoms 'symb_sim' der leeren Liste wie im 1. Fall der Abarbeitung. Dann ist mit 'ende(*Aktiv1, EndLokationen*) zu überprüfen, ob die aktive Lokation einer Endlokation von *HybSys* entspricht. Das Zielatom 'ende' wird zum Constraintatom 'inlist' entfaltet. Kann 'inlist' zu 'true' ausgewertet werden, so befindet sich die Ausführung in einem erfolgreichen Endzustand, anderenfalls scheitert die Ausführung durch die Vereinfachung im Constraintspeicher und befindet sich in einem erfolglosen Endzustand.


⟨query(Eigenschaft, Hybrides System, Startlokation, Startzeit, StartVarBelegung) , true⟩
Entfalten
⟨include(HybSys) ∧ symb_sim(Zeitwort, Aktiv_Lok, Tstart, VARstart) ,
query(Zeitwort, HybSys, Aktiv_Lok, Tstart, VARstart)
≙
query(Eigenschaft, Hybrides System, Startlokation, Startzeit, StartVarBelegung) ∧ true⟩
Vereinfachen
include(HybSys) → true
1. Aufruf von symb_sim :
⟨symb_sim(Zeitwort, Aktiv_Lok, Tstart, VARstart) , true ∧ Zeitwort ≙ Eigenschaft ∧ . . . ⟩
1. Fall
Entfalten
2. Fall
⟨ende(Aktiv1, EndLokationen) ,
symb_sim([], Aktiv1, Tvor, VARvor)
≙
symb_sim(Zeitwort, Aktiv_Lok, Tstart, VARstart)
∧ true ∧ Zeitwort ≙ Eigenschaft ∧ . . . ⟩
Entfalten
⟨inlist(Aktiv2, D) ,
ende(Aktiv2, endlokationen(D))
≙
ende(Aktiv1, EndLokationen) ∧ Zeitwort ≙ [] ∧ . . . ⟩
inlist(Aktiv2,D)
→ true
Vereinfachen
inlist(Aktiv2,D)
→ false
⟨⊤, true ∧ Aktiv1 ≙ Aktiv2 ∧ . . . ⟩
Erfolg
⟨⊤, false ∧ Aktiv1 ≙ Aktiv2 ∧ . . . ⟩
Scheitern
⟨combi_active_trans(⟨Symbole, T, [VAR, VARnach], Beding⟩,
Aktiv1, Nachf_Lok, Tvor, VARvor) ∧
symb_sim(Restwort, Nachf_Lok, T, VARnach),
symb_sim([⟨Symbole, T, [VAR, VARnach], Beding⟩ Restwort],
≙ Aktiv1, Tvor, VARvor)
symb_sim(Zeitwort, Aktiv_Lok, Tstart, VARstart) ∧
true ∧ Zeitwort ≙ Eigenschaft ∧ . . . ⟩
Entfalten
⟨synchron(Symbol1, VAR1, VARnach1, Aktiv2, SynBeding,
SynAktion) ∧
release(Symb1, T1, Tvor1, VAR1, VARvor1, Aktiv2,
Nachf2, Aktivitäten, UnbedingtTrans,
BedingtTrans) ∧
conclude(Symb1, VAR1, VARnach1, Nachf1, Aktionen,
EintrittInv) ∧
symb_sim(Restwort, Nachf_Lok, T, VARnach) ,
combi_active_trans(⟨Symb1, T1, [VAR1, VARnach1], Bed1⟩,
≙ Aktiv2, Nachf1, Tvor1, VARvor1)
combi_active_trans(⟨Symbole, T, [VAR, VARnach], Beding ⟩,
Aktiv1, Nachf_Lok, Tvor, VARvor)
∧ true ∧ Zeitwort ≙ [⟨Symbole, T, [VAR, VARnach], Beding⟩
Restwort] ∧ . . . ⟩
Entfalten

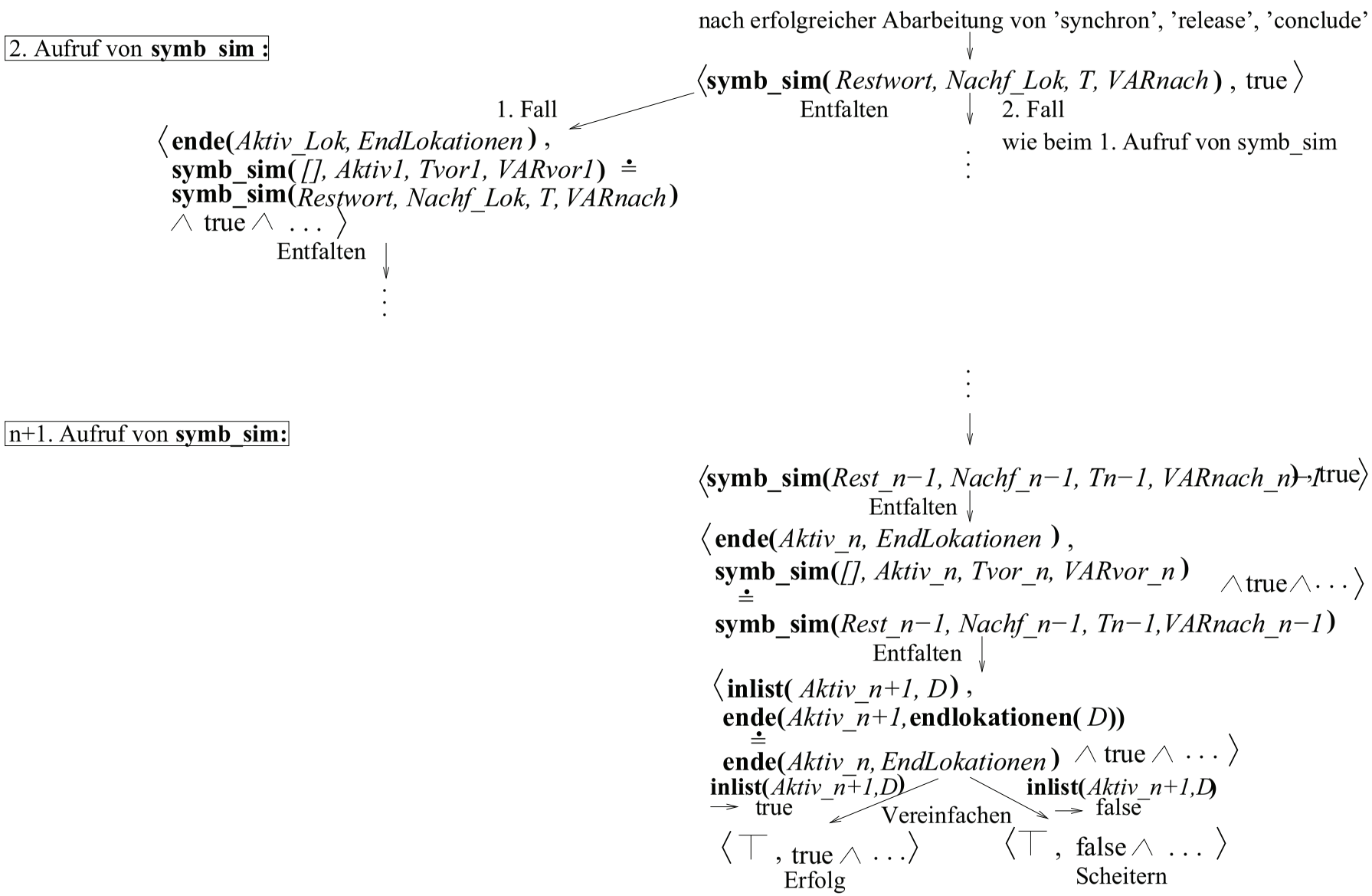


Abbildung 5.1: Top-Down Ausführung der Symbolischen Simulation

Ist das vorliegende Zeitwort zur symbolischen Simulation nicht leer, so erfolgt eine Entfaltung. Für das erste Tupel des Zeitwortes ⟨*Symbole, T, [VAR, VARnach], Beding*⟩ werden mit 'combi_active_trans' bezüglich aller aktiven Transitionen und Verbindungen, die mit den Symbolen des Tupels in Zusammenhang stehen, Bedingungen über der Zeit *T* und den Variablen *VAR* bzw. *VARnach* gesammelt und konjunktiv verknüpt. Weiterhin wird die Nachfolgelokation *Nachf_Lok* zur Abarbeitung des verbleibenden Wortes *Restwort* ermittelt. Die ererbten und synthetisierten Parameter sind in 'combi_active_trans' und 'symb_sim' fett hervorgehoben. Das Zielatom 'combi_active_trans' lässt sich weiter zu 'synchron', 'release' und 'conclude' entfalten.
Hier sind noch einmal die ererbte Bedingung *Beding* und die synthetisierten Bedingungen *SynBeding*, *SynAktion*, *Aktivitäten*, *UnbedingtTrans*, *BedingtTrans*, *Aktionen* und *EintrittsInv* fett hervorgehoben. Stellt der Constraintlöser fest, dass all diese Bedingungen untereinander konsistent sind, so können 'synchron', 'release' und 'conclude' erfolgreich abgearbeitet werden.
Nach der erfolgreichen Abarbeitung erfolgt ein 2. Aufruf von 'symb_sim', welcher sich nur noch auf das verbleibende Wort *Restwort* bezieht. Durch die aus dem 1. Aufruf übernommene Zeit *T* als *Tvor* und die übernommenen Variablen *VARnach* als *VARvor* für den 2. Aufruf, sind auch die an diese Parameter gebundenen Bedingungen aus dem 1. Aufruf für die weitere Konsistenzprüfung relevant. Die im 1. Aufruf erläuterten zwei Fälle werden auch hier unterschieden. Doch das erfolgreiche Abarbeiten des 2. Tupels des Zeitwortes hängt nun von der Konsistenz der Bedingungen aus dem 1. Aufruf mit den neu berechneten Bedingungen des 2. Aufrufes ab. Hat das Zeitwort eine Länge von n Tupeln, so werden für jeden der ersten n Aufrufe die zwei Fälle unterschieden. Für den Erfolg der weiteren Abarbeitung ist zusätzlich zu den in diesem Aufruf berechneten Bedingungen die gesamte Historie der in allen vorangegangenen Aufrufen ermittelten Bedingungen verantwortlich. Genauer werden diese Bindungen für die Abbildung 5.2 beschrieben. Den Abschluss bildet der n+1. Aufruf, in welchem ein leeres Wort vorliegt. Das gesamte Zeitwort gilt als akzeptiert, wenn die Lokation 'Aktiv_Lok' im n+1. Aufruf von 'symb_sim' eine Endlokation ist.
Mit der Abbildung 5.2 werden die Beziehungen zwischen den Variablen eines Aufrufes von 'symb_sim' untereinander und den Variablen nachfolgender Aufrufe zueinander graphisch dargestellt. Weiterhin wird die Bindung einzelner Bedingungen an die Variablen als eindeutige Abbildungen wie 'aktivitäten', 'aktionen' und 'synAktion', sowie mehrdeutiger Abbildungen wie 'beding', 'synBeding', 'unbedingtTrans', 'bedingtTrans' und 'eintrittInv' veranschaulicht. Variablen und Bedingungen, die zu ein und demselben Aufruf gehören, sind mit der Nummer des Aufrufes indiziert. Durchgehende Pfeile kennzeichnen Abhängigkeiten zwischen Variablen und gestrichelte Pfeile Bindungen von Bedingungen an die jeweiligen Variablen. Mit 'T[i]', i 0,...,n, sind die Zeiten des Auftretens der Symbole eines Tupels bezeichnet, mit 'VAR[i]' und 'VARnach[i]' die jeweiligen Variablenbelegungen bezüglich dieser Zeiten zum Auslösen und Abschließen der zugehörigen Transition im hybriden System. Mit 'beding[i]' sind Bedingungen gegeben, die der Nutzer an das Zeitwort stellen kann. Alle anderen Bedingungen ergeben sich aus dem hybriden System. Diese Bedingungsarten, welche einander gegenüber stehen, sind wäh-

rend der symbolischen Simulation auf Konsistenz zu überprüfen. Aus dieser Sichtweise ergeben sich zum Beispiel Anfragemöglichkeiten wie:

- Falls alle 'beding[i]' mit 'true' belegt sind, so stellt sich die Frage, ob und unter welchen Bedingungen an die Zeiten 'T[i]' und Variablenbelegungen 'VAR[i]' Zeitwörter vom hybriden System akzeptiert werden.

- Falls 'beding[i]' existieren, welche die Zeiten 'T[i]' und Variablenbelegungen 'VAR[i]' zusätzlich durch den Nutzer einschränken, so stellt sich die Frage, ob und in welchem Maße gegebene Zeitwörter vom hybriden System akzeptiert werden.

Genauere Betrachtungen dazu sind noch einmal im Abschnitt 7.2 aufgeführt.

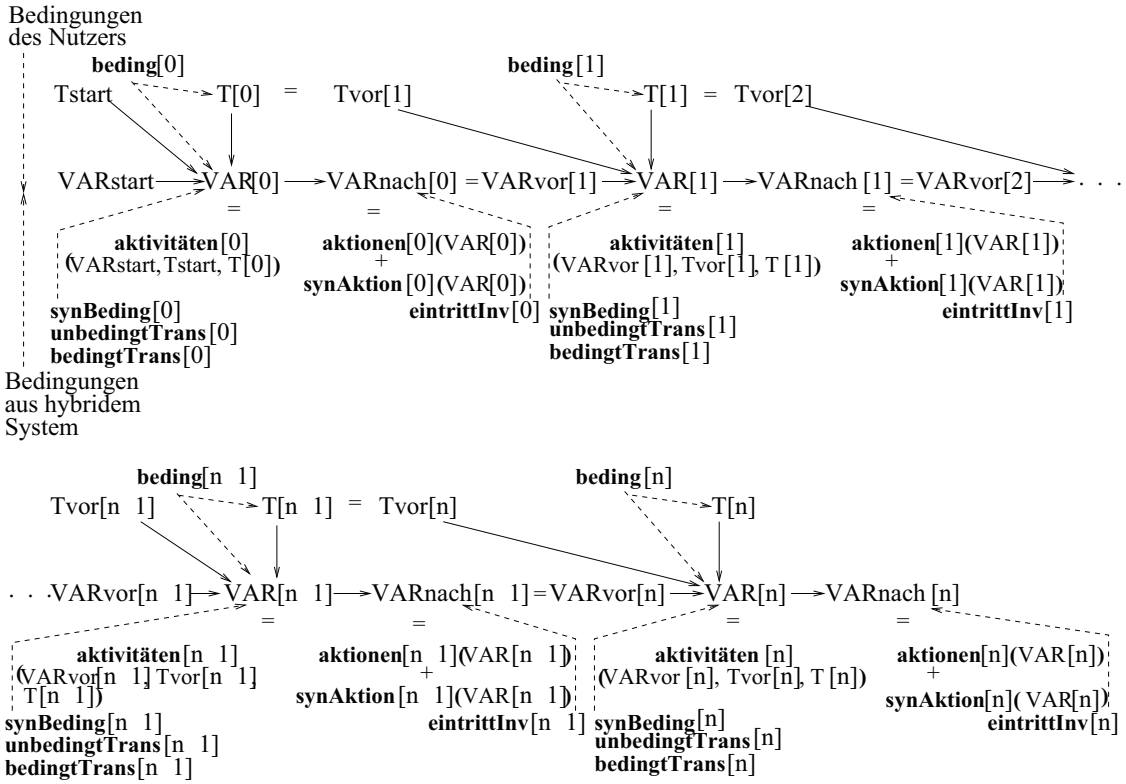


Abbildung 5.2: Abhängigkeiten von Variablen und Bindung von Bedingungen

Die Ausführung der Abbildung 5.2 lässt sich folgendermaßen interpretieren:

Gesuchte Größen: Belegungen und Bedingungen an alle 'T[i]' und 'VAR[i]', so dass ein gegebenes Zeitwort akzeptiert wird
Gegebene Größen: Belegung für 'Tstart' und 'VARstart', Bedingungen 'beding[i]' des Nutzers,

Bedingungen des hybriden Systems:
Funktionen ’aktivitäten’, ’aktionen’ und ’synAktion’,
Relationen ’beding’, ’synBeding’,’unbedingtTrans’,
’bedingtTrans’ und ’eintrittInv’

Ablauf:
1. Aufruf:

1. Sämtliche Variablenwerte aus ’VAR[0]’ sind über die Aktivitäten in Abhängigkeit von ’Tstart’, ’VARstart’ und ’T[0]’ bestimmt. Ist die aktive Lokation zu Beginn der symbolischen Simulation eine Anfangslokation, so ergibt sich ’T[0]’ aus ’Tstart’ und ’VAR[0]’ aus ’VARstart’.

2. An die Variablenwerte aus ’VAR[0]’ und die Zeit ’T[0]’ wird die Bedingung ’beding[0]’ gebunden. Weiterhin erfolgt eine Bindung der Bedingungen ’synBeding[0]’, ’unbedingtTrans[0]’ und ’bedingtTrans[0]’ an die Variablenwerte aus ’VAR[0]’.

3. Die Variablenwerte aus ’VARnach[0]’ sind über die Aktionen der aktiven Transitionen ’aktionen[0]’ und die Aktionen der Synchronisationsverbindungen ’synAktion[0]’ in Abhängigkeit von ’VAR[0]’ festgelegt.

4. An die Variablenwerte aus ’VARnach[0]’ wird die Bedingung ’eintrittInv[0]’ gebunden.

5. Die Zeit ’T[0]’ wird dem nachfolgenden Aufruf als ’Tvor[1]’ zur Verfügung gestellt und die Variablenbelegung ’VARnach[0]’ als ’VARvor[1]’.

Für jeden weiteren i-ten Aufruf gilt:

1. Sämtliche Variablenwerte aus ’VAR[i]’ sind über die Aktivitäten in Abhängigkeit von ’Tvor[i]’, ’VARvor[i]’ und ’T[i]’ bestimmt.

2. Weiterer Ablauf äquivalent zum 1. Aufruf ...

Mit jeder Bindung einer Bedingung an die Variablen erfolgt ein erneuter Test auf Konsistenz. Über die Abhängigkeiten der Variablen werden Bindungen an Bedingungen weitergereicht, welche somit zur Konsistenzprüfung aller nachfolgenden Aufrufe von Bedeutung sind und umgekehrt rückbezüglich Auswirkungen auf die vorherigen Ausrufe hat wie in Abbildung F.1 des Abschnittes F.2 des Anhangs F zu sehen ist. Zusätzlich ist in demselben Abschnitt noch einmal die umgekehrte Abhängigkeit der Zeiten ’T[i]’ von ’Tvor[i]’, ’VARvor[i]’ und ’VAR[i]’ dargestellt, um zu verdeutlichen, dass Bedingungen, die aus dem hybriden System auf die Variablen Einfluss haben, auch die Zeiten zur Akzeptanz der einzelnen Tupel beeinflussen.

5.3.4 Symbolische Simulation und Bounded Model Checking

Bevor die symbolische Simulation an einem Beispiel nachvollzogen wird, soll hier der enge Zusammenhang dieser Simulation zum Bounded Model Checking herausgearbeitet werden. Auf diese Art und Weise wird neben dem Bezug zur diskret-ereignisorientierten Simulation die Verbindung zur automatischen Verifikation geschaffen.
Bereits in [ACKS02] wurde Bounded Model Checking für Systeme bezüglich zeitlicher Betrachtungen und in [Sor02, WZP03] für Zeitautomaten formalisiert. Für hybride Systeme [FH05, HEFT08, EFH08] wurde Bounded Model Checking auf SAT Basis in Verbindung mit linearer Programmierung [WW99] und SAT Modulo Theorien mit ODEs bzw. in [ÁBKS05] auf SAT Basis in Verbindung mit domain-spezifischen Lösern für den Bereich der reelen Zahlen realisiert. Vorgehensweisen wie bei der symbolischen Simulation im Hardwarebereich [CCK04, CCK07] sowie bei der gebundenen, pfad-orientierten Erreichbarkeitsanalyse für hybride Systeme [LAB07] zur Lösung von Problemen des Bounded Model Checking sind richtungsweisend für unsere Methode der symbolischen Simulation. Das Ziel unseres Ansatzes besteht im Gegensatz zu [CCK04, CCK07] in der Analyse von Softwaresystemen, welche zeitliche Bedingungen in Zusammenhang mit Bedingungen für sich kontinuierlich ändernde Umgebungsgrößen erfüllen müssen. Doch wie z.B. in [CCK04] nach der Reparametrisierung, die der Verkürzung von zu berechnenden Funktionen des SAT Problems dient, kein Gegenbeispiel ohne Speicherung des vorhergehenden Zustandes der Funktionen erstellbar ist, so muss auch in unserem Ansatz vor der Vereinfachung von Constraints der vorliegende Constaintspeicher im Gedächtnis erhalten, um später im Fall eines Widerspruches ein aussagekräftiges Gegenbeispiel erstellen zu können. In der Arbeit von [LAB07] werden für Eigenschaften interessante Pfade auf Basis der linearen Programmierung ausgewertet. In unserem Ansatz werden Pfade als Zeitwörter interpretiert, deren Akzeptanz durch die Ausführung in CLP überprüft wird. Auf diese Art und Weise besitzt unsere Methode einen Zugang zum Bereich der formalen Sprachen, welcher Rückschlüsse auf diesen Bereich zulässt bzw. Erkenntnisse aus dem Bereich der Analyse hybrider Systeme in den Bereich der formalen Sprachen integrieren lässt. In zukünftigen Arbeiten wird CLP auf der einen Seite für unsere symbolische Simulation genutzt, kann aber auch durch Erkenntnisse und Erfahrungen aus dem Bereich des SAT-basierten Bounded Model Checking erweitert werden. Dabei sei insbesondere auf domänenspezifische, konfliktlernende und backtracking-beschleunigende Methoden aus [FH05], wie in Abschnitt 5.2.2 aufgeführt, verwiesen.
Unterschiede des Bounded Model Checking ergeben sich in verschiedenen Systemen durch das Auftreten unterschiedlicher Klassen von Bedingungen, wodurch unterschiedliche domain-spezifische Constraintlöser wie in [ÁBKS05] zum Einsatz kommen. Im Zusammenhang mit den unterschiedlichen Klassen von Bedingungen ergeben sich weiterhin unterschiedliche Repräsentationen und Interpretationen der Transitionsrelation des Modells und Spezifikationen der Eigenschaften für das Bounded Model Checking. Wie in Abschnitt 5.2.2 zum Bounded Model Checking erwähnt, besteht ein *Entwurf* aus einem *gegebenen Modell*, einer *zu prüfenden Eigenschaft* und einem *Übersetzungsschemata* in eine *zu prüfende Formel*. Aufgrund der Unterschiede der zugrundeliegenden Modelle und

Spezifikationen von Eigenschaften entstehen Unterschiede in der Repräsentation der zu prüfenden Formel und der zur Lösung diese Formel eingesetzten Verfahren.
Gemeinsamkeiten bestehen in den 3 Schritten der Übersetzung, um das SAT-basierte Bounded Model Checking auszuführen:

1. Übersetzung gültiger Modellpfade mit festgesetzter Grenze k in aussagenlogische Formel MF,
2. Übersetzung der Eigenschaft in aussagenlogische Formel EF einschließlich zugehöriger Schleife l und Grenze k,
3. Bilden der zu prüfenden Formel $PF \quad MF \wedge EF$.

In Abhängigkeit vorliegender Modelle und Eigenschaften können die aussagenlogischen Formeln MF und EF dabei mit zusätzlichen Bedingungen in Form arithmetischer Ausdrücke, Differentialgleichungen usw. wie bei den SMT (SAT Modulo Theorien) [NOT04, EFH08, EKKT08, FNORC08] verbunden werden. In Tabelle 5.1 sind verschiedene Modelle und Eigenschaften, die die Grundlage von SAT-basiertem Bounded Model Checking bilden, zusammengefasst.

Modell	*Eigenschaft*	*Übersetzung*
Kripke Struktur [BCCZ99]	LTL Formel	SAT Formel
Zeitautomat (überführt in Kripke Struktur) [ACKS02]	LTL Formel	Formel für MathSAT (reele Variablen bilden temporalen Teil des Zustandsraumes und dessen Entwicklung)
Zeitautomat [Sor02]	LTL Formel mit Bedingungen an Uhren	Boolsche Constraint-formel verbunden mit arithmetischen Constraints
Zeitautomat (überführt in diskretisierten, erweiterten Regionen-automat) [WZP03]	Erreichbarkeits-eigenschaft	SAT Formel
linear hybride Automaten (lineare, null-eins Constraints boolscher Werte für diskrete und überwachte,	Konjunktion aus linearen, null-eins und überwachten, linearen Constraints	Constraints für Anfangs-zustände + alternierende Transitionsfolge aus linearen, null-eins und überwachten, linearen Constraints

Tabelle 5.1 siehe nächste Seite

Tabelle 5.1 siehe vorige Seite

Modell	*Eigenschaft*	*Übersetzung*
lineare Constraints reeler Werte für kontinuierliche Zustandskomponenten) [FH05]		im Zusammenhang mit Zustandsinvarianten und Aktivitäten (gelöst durch SAT Prozedur + lineare Programmierung)
linear hybride Automaten [ÁBKS05]	Konjunktion aus Sprung- und Flussbedingungen	Formel aus Sprung- und Flussbedingungen über einerseits Zustandsvariablen und andererseits reellwertigen Variablen zur Beschreibung der Zeitdauer des Flusses (gelöst durch SAT Prozedur, die domain-spezifische Constraintlöser aufruft)
linear hybride Automaten (Pfade in Form linearer Programme) [LAB07]	Erreichbarkeits-eigenschaft	Programm linearer Un-(Gleichungen)

Tabelle 5.1: Grundlagen des SAT-basierten BMC

In unserem Ansatz wird ein gültiger Modellpfad MP in einem hybriden System bis zur Grenze k, die die Anzahl ausgeführter Schritte zur Akzeptanz eines Zeitwortes darstellt, durch die Formel $[\![MP]\!]_k$ bedingt, wobei gilt:

$$[\![MP]\!]_k = \bigwedge_{i=0}^{k} \begin{array}{l} (receive_i(RSymbole_i) \wedge send_i(SSymbole_i) \wedge synBeding_i \wedge \\ synAktion_i \wedge aktivitäten_i \wedge unbedingtTrans_i \wedge \\ bedingtTrans_i \wedge aktionen_i \wedge eintrittInv_i) \end{array}$$

Die Formel $[\![MP]\!]_k$ stellt eine Folge aufeinanderfolgender Transitionen t_i mit $i \in \{0, .., k\}$ in einem flachen hybriden Automaten A dar, die wie in vorhergehenden Abschnitten durch die Prädikate:

$receive_i(RSymbole_i)$	Erhalt zu empfangender Symbole,
$send_i(SSymbole_i)$	Sendung zu sendender Symbole,
$synBeding_i$	Erfüllung der Systembedingung,
$synAktion_i$	Erfüllung der Systemzuweisung,
$aktivitäten_i$	Erfüllung der Aktivitäten,
$unbedingtTrans_i$	Erfüllung der transformierten Invariante der Ausgangslokation,

$bedingtTrans_i$	Erfüllung der Transitionsbedingung,
$aktionen_i$	Erfüllung der Transitionszuweisung und
$eintrittInv_i$	Erfüllung der Invariante der erreichten Lokation

gekennzeichnet sind und wobei gilt:

1. Die Bedingung $unbedingtTrans_0$ wurde aus der standardmäßig gesetzten Invariante der Anfangslokation von A erzeugt.
2. Alle weiteren Bedingungen $unbedingtTrans_i$ mit $i \in \{1, .., k\}$ ergeben sich aus $eintrittInv_{i-1}$.
3. Die Transition t_k führt zu einer Endlokation.

Eine Eigenschaft wird durch die Formel $[\![ZW]\!]_k$ beschrieben, welche aus einem Zeitwort ZW hergeleitet wurde, das von einem Nutzer durch Bedingungen $beding_i$ eingeschränkt sein kann und deren k Tupel im hybriden System auf Akzeptanz mit dem Erreichen einer Endlokation überprüft werden sollen. Dabei gilt:
$[\![ZW]\!]_k = \bigwedge_{i=0}^{k} \langle Symbole_i, T_i, VAR_i, beding_i \rangle$, welche besagt, die Symbole $Symbole_i$ sollen zu einer symbolischen Zeit T_i unter der Variablenbelegung VAR_i und der Einhaltung der Nutzerbedingung $beding_i$ akzeptiert werden.
Die zu prüfende Formel $[\![PF]\!]_k$ ergibt sich aus:
$[\![PF]\!]_k = [\![MP]\!]_k \wedge [\![ZW]\!]_k$, wobei gelten muss $Symbole_i \quad RSymbole_i \cup SSymbole_i$ für $i \in \{0, .., k\}$.
Die Formel $[\![PF]\!]_k$ wird dabei durch die Anfrage der als Zeitwort vorliegenden Eigenschaft an ein als CLP Programm beschriebenes, hybrides System wie in Abbildung 5.1 überprüft.

5.3.5 Beispiel der symbolischen Simulation

Als Beispiel ist der Ablauf und dessen Ergebnisse für eine Anfrage nach dem Zeitwort aus dem Beispiel 5.3.1 an das hybride System des Studienbüros mit dem Prüfungsablauf aus Abbildung 4.11 tabellarisch in den Abbildungen 5.3 und 5.4 aufgeführt.
In Abbildung 5.3 sind die Namen der Variablen an die im Algorithmus der symbolischen Simulation verwendeten Namen angepasst worden. Von der Seite des Nutzers wurden keine zusätzlichen Bedingungen gestellt, so dass 'beding' in allen 3 Fällen mit 'true' belegt ist. Die Funktionen 'aktivitäten' 'synAktion' und 'aktionen' sowie Relationen 'synBeding', 'unbedingtTrans', 'bedingtTrans' und 'eintrittInv' ergeben sich aus dem hybriden System 'Pruefung_Bearbeitung'. Durch die Umrahmung wurde die Zusammengehörigkeit von Constraints gekennzeichnet. Die gestrichelten Linien verdeutlichen noch einmal die Abhängigkeit der Variablenbelegung zu einer Zeit 'T[i]' von der Variablenbelegung beim Abschluss einer Transition zur Zeit 'T[i-1]'.
Unter der Voraussetzung:

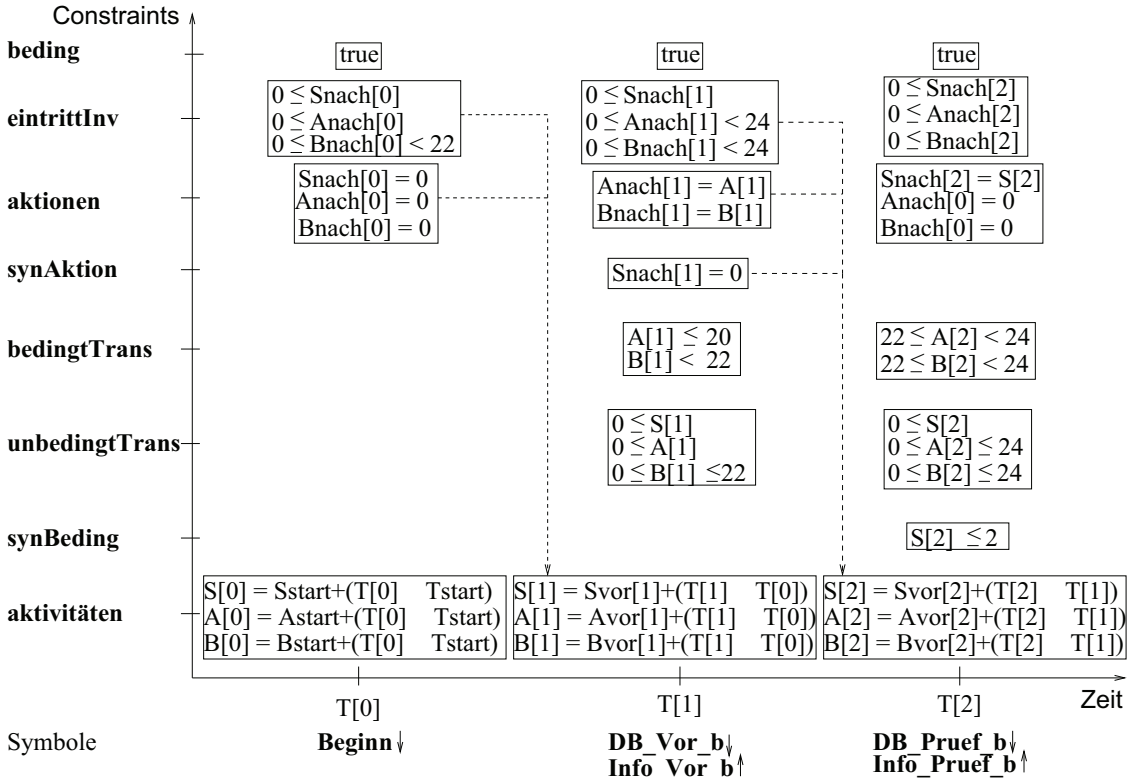


Abbildung 5.3: Lauf mit zugehörigen Constraints

- Tstart 0,
- ’VARstart [Sstart, Astart, Bstart] [-1,-1,-1] und
- aktive Lokation zu Beginn des Laufes ’Aktiv_Lok’ Anfangslokation

ergeben sich während der Ausführung der symbolischen Simulation durch die Bedingungen aus Abbildung 5.3 Ergebnisse, wie diese in der Abbildung 5.4 angegeben sind.
Abhängigkeiten von Constraints, die zur Vereinfachung und Ersetzung in Bedingungen führen, können über die durchgehenden Pfeile nachvollzogen werden. Schlussfolgerungen, die sich durch Vereinfachungen von Constraints ergeben, sind nicht umrahmt. Ergebnisse für die Zeiten der Akzeptanz von Symbolen wurden unterstrichen. Die Zeit ’T[0]’ zur Akzeptanz des Symbols ’Beginn’ ist durch ’Tstart’ gleich Null. Die Werte von ’S[0]’,’A[0]’ und ’B[0]’ ergeben sich aus ’Sstart’, ’Astart’, und ’Bstart’. Beim ersten Aufruf der symbolischen Simulation mit dem Prädikat ’symb_sim’ ergibt die Vereinfachung der Constraints der Aktionen und der Invarianten zum Eintreten in die nachfolgende Lokation die Bedingungen ’Snach[0] 0’,’Anach[0] 0’ und ’Bnach[0] 0’.
Aus dieser Schlussfolgerung und dem Ergebnis ’T[0] 0’ errechnet sich die Variablenbe-

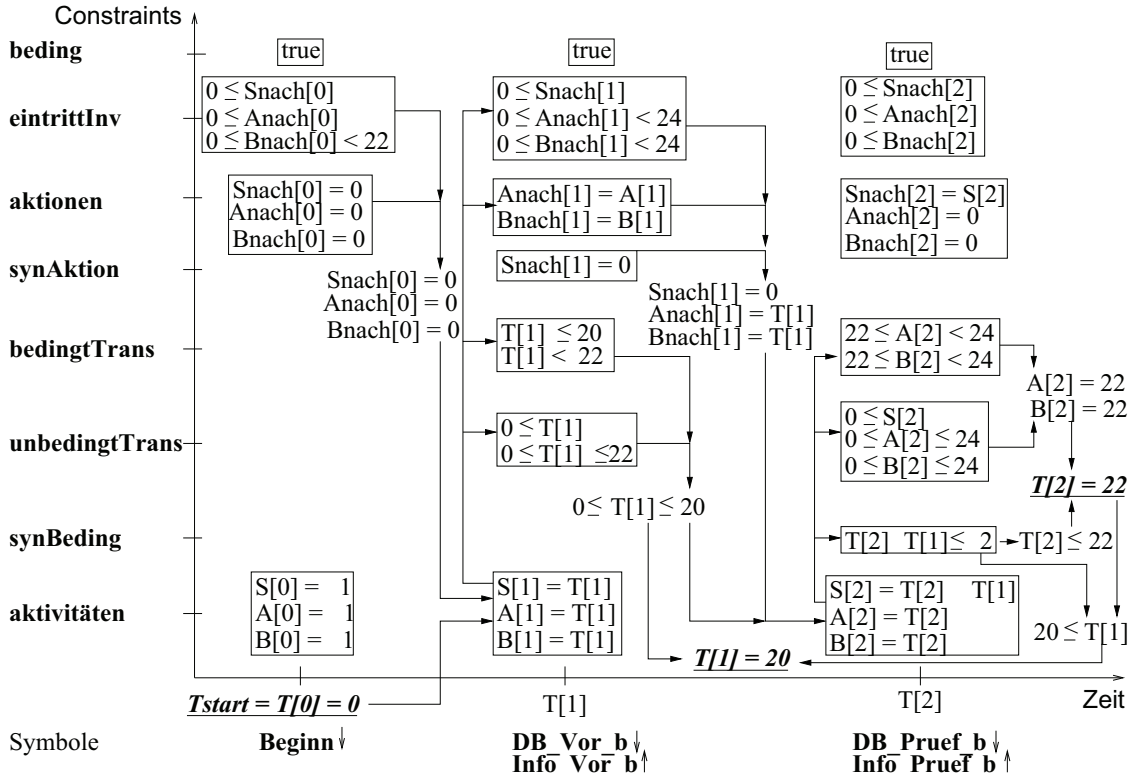


Abbildung 5.4: Ergebnisse für Zeiten und Variablenbelegungen

legung zur Akzeptanz der Symbole 'DB_Vor_b ↓' und 'Info_Vor_b ↑', wobei die Constraints 'S[1] T[1]', 'A[1] T[1]' und 'B[1] T[1]' gelten. Dieses Ergebnis besitzt einen Einfluss auf alle weiteren Constraints bei der Abarbeitung des zweiten Aufrufes von 'symb_sim'. Für die Aktionen dieses Aufrufes sei hervorgehoben, dass sich die Belegung der Variable 'Snach[1] 0' aus der Aktion einer Synchronisationsverbindung ergibt und 'Anach[1]' sowie 'Bnach[1]' den Wert von 'A[1]' sowie 'B[1]' übernehmen, da explizit keine Aktionen für diese Variablen angegeben wurden. Durch die Schlussfolgerungen '0 ≤ T[1] ≤ 20' und 'Snach[1] 0','Anach[1] T[1]' und 'Bnach[1] T[1]' wird die Berechnung der Aktivitäten des dritten Aufrufes beeinflusst.
Zur Akzeptanz der Symbole 'DB_Pruef_b ↓' und 'Info_Pruef_b ↑' wurde die Bedingung 'S[2] ≤ 2' an der zugehörigen Synchronisationsverbindung ermittelt. Die Ersetzung dieses Constraints zu 'T[2]-T[1] ≤ 2' und die berechnete Bedingung '0 ≤ T[1] ≤ 20' im zweiten Aufruf führt zur Schlussfolgerung 'T[2] ≤ 22'. Die Bedingung 'T[2] 22' ergibt sich aus den Aktivitäten und Bedingungen für 'A[2]' und 'B[2]' in Verbindung mit dieser Schlussfolgerung. In Verbindung mit dem Constraint 'T[2]-T[1] ≤ 2' entsteht jedoch dadurch rückwirkend die Forderung, dass '20 ≤ T[1]' ist, was bezüglich des Constraints '0 ≤ T[1] ≤ 20' aus dem zweiten Aufruf zur Folge hat, dass 'T[1] 20' sein muss.
Zusammenfassend gilt unser gewähltes Beispielwort als akzeptiert, wenn:

- die Voraussetzung zur Prüfung in der 20. Zeiteinheit (Monat) abgelegt wurde und
- die Prüfung in der 22 Zeiteinheit.

Die verstärkte Bedingung 'X $\leq$ 20' des Verwaltungsprozesses der Abbildung 4.10 im Studienbüro führt zu einer zeitlichen Einschränkung des Ablegens der Voraussetzung. Die zusätzliche Bedingung 'S $\leq$ 2' im SHA für das Studienbüro und den Prüfungsablauf, die durch eine übergeordnete Verordnung festgelegt wurde, beschränkt die Zeiträume für die Voraussetzung und die Prüfung so stark, dass zum Ablegen nur noch eine Zeiteinheit zur Verfügung steht. Dieses Ergebnis ist nicht trivial aus dem hybriden System der 'Pruefung_Bearbeitung' zu erkennen.

5.3.6 Stand der symbolischen Simulation

Die symbolische Simulation ist zur Zeit auf eine Teilmenge der mit MODEL-HS und VYSMO modellierbaren hybriden Systeme beschränkt, da folgende theoretische und praktische Probleme bestehen, die beachtet bzw. gelöst werden müssen. Aus entwurfstechnischer Sicht ist es für unsere Sprachen MODEL-HS und VYSMO zur Schaffung der Übersichtlichkeit und guten Wiederverwendung wichtig, die beliebige Schachtelung von HHAs und SHAs als SHHAs zuzulassen. Dabei können sich verschiedene HHAs bzw. SHAs in Bibliotheken befindliche Automaten teilen. Zur Modellierung ist die in [AKY99] definierte Wohlstrukturiertheit in unseren hierarchisch hybriden Automaten aufgehoben. Angelehnt an [AKY99] gilt der folgende Begriff.

Begriff 5.3.8
Ein hierarchisch hybrider Automat HHA heißt **wohlstrukturiert**, wenn sich HHA ausschließlich auf seiner obersten Hierarchiestufe mit anderen Automaten synchronisiert.

Das heißt, trotz der Aufblähung durch zu kopierende Signale sind in den Modellen zur Darstellung praktisch möglicher Fälle der Synchronisation mit der Umwelt HHAs zulässig, die sich auch aus unteren Hierarchiestufen mit anderen Automaten synchronisieren. Zur symbolischen Simulation sind folgende Entscheidungen getroffen worden:

1. Die Produktbildung ist bezüglich nebenläufiger und sequentieller Operatoren assoziativ. Deshalb werden SHAs auf SHAs zu einem SHA zusammengefasst.
2. Ein SHA kann ausschließlich auf der obersten Hierarchieebene auftreten und HHAs beinhalten. Komplexe Lokationen der hierarchisch hybriden Automaten basieren dabei wieder auf hierarchisch hybriden Automaten.
3. HHAs sollen wohlstrukturiert sein.
4. HHAs werden vor der Ausführung der symbolischen Simulation in flache Automaten transformiert.

Wie auf den folgenden Seiten in der Tabelle 5.2 zu sehen ist, existieren nur ausgewählte Klassen hybrider Systeme, die theoretisch entscheidbar sind. Praktisch ist die Entscheidbarkeit auch in Bezug auf eine modulare Struktur zu bewerten. Hier wird gezeigt, dass der Algorithmus zur symbolischen Simulation auf der Basis der transitiven Hülle für die Berechnung der Transitionen eines SHA im schlechtesten Fall exponentiell ist. Die Berechnung der transitiven Hülle eines hierarchisch hybriden Automaten HHA bezüglich einer gegebenen Transition gibt die Menge aller hierarchisch hybriden Automaten zurück, die in einem synchronisierend hybriden Automaten mit dem Automaten HHA über direkte und durch Transitivität entstandene Synchronisationswege verbunden sind. Dabei werden alle Signale, die Bedingung und die Zuweisungsmenge für die Verbindung mit allen Automaten ermittelt. Zur Vereinfachung wird von den Annahmen ausgegangen:

- Jeder Automat ist nur mit einer Transition an einer Synchronisationsverbindung beteiligt ist.

- Zwischen Automaten, die bereits mit HHA eine Synchronisationsverbindung besitzen sind keine weiteren Synchronisationsverbindungen zulässig. In Abbildung 5.5 sind unzulässige Verbindungen mit einem Kreuz gekennzeichnet. Diese Annahme führt zu einer Einschränkung der praktisch möglichen Problemstellungen, ändert jedoch nichts an der Komplexität des Algorithmuses.

In Erinnerung an die Funktion 'transitive_closure' des Abschnittes 4.4.2 und mit Bezug auf die vollständige Umsetzung dieser Funktion im Anhang C berechnet die Funktion 'synchron' alle direkten Synchronisationsverbindungen bezüglich einer in einem 'HHA' auftretenden Transition. Die Funktion 'new_diff' gibt die Menge aller 'HHA_i', i 1,...,n, zurück, die in Verbindung mit dem 'HHA' stehen und berechnet gleichzeitig fortlaufend aus den in Abbildung 5.5 befindlichen Symbolen GR[HHA_i] und AS[HHA_i] die Gesamtmenge der zur Synchronisation gehörenden Symbole, aus den Bedingungen 'TrCond_i' und 'SyCond_i' die Gesamtbedingung sowie aus den Zuweisungsmengen 'TrAssign_i' und 'SyAssign_i' die Zuweisungsmenge für die gesamte Synchronisation. Die Funktion 'for_all_diff' ermöglicht die Berechnung der direkten Synchronisationsverbindungen aller Automaten 'HHA_i' mit der gleichen Pfadlänge eines Baumes wie in Abbildung 5.5. Die Funktion 'transitive_closure' berechnet die Synchronisationsverbindungen für alle Tiefen des Baumes bis zu einem Fixpunkt, der durch die endliche Menge an Synchronisationsverbindungen zwischen der endlichen Menge an Automaten gegeben ist. Angenommen, maximal kann jeder hierarchisch hybride Automat 'n' direkte Synchronisationsverbindungen zu weiteren Automaten bezüglich einer Transition aus 'HHA' besitzen und der längsten Pfad des Baumes wie in Abbildung 5.5 ist hat eine Länge 'l', so arbeitet der Algorithmus zur Berechnung der transitiven Hülle bezüglich der Länge 'l' im schlechtesten Fall exponentiell mit einer Komplexität von $O(n^l)$.
Theoretisch ist es möglich, diese Komplexitätsklasse beizubehalten, wenn SHAs auf weiter unten liegenden Hierarchieebenen verwendet werden, ohne explizit deren Produkt zu bilden.

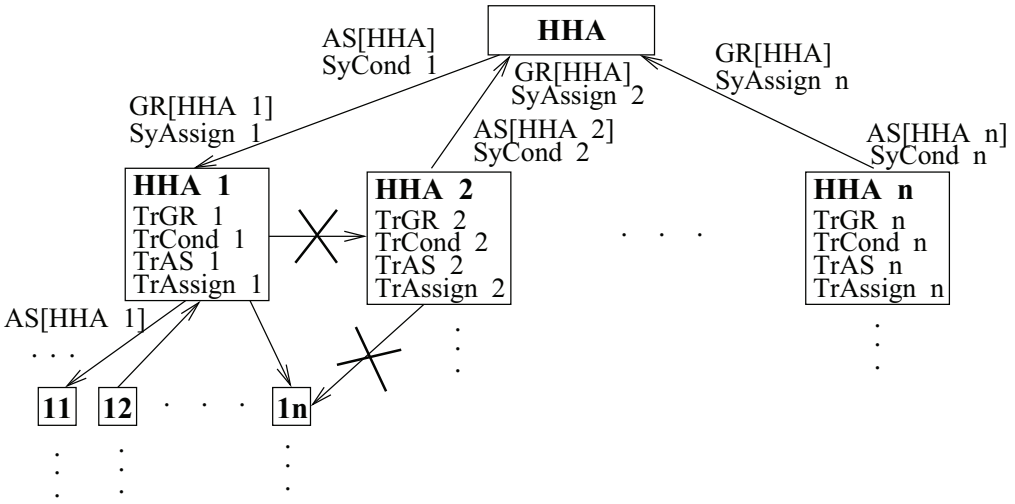


Abbildung 5.5: Synchronisationsverbindungen

Dann wird, wie in Abbildung 5.6 auf jedem SHA während der symbolischen Simulation die Berechnung der transitiven Hülle mit der Komplexität $O(n^l)$ angewandt. Maximal seien in einem gegebenen SHA weitere 'm' Unter-SHA an der Synchronisation bezüglich einer Transition beteiligt, in denen wieder 'm' SHA zur Synchronisation beitragen können. Der längste Pfad soll mit der maximalen Tiefe 'd' bezeichnet werden. Die Komplexität der gesamten Berechnung ist dann $O(m^d \cdot n^l)$.
Aus technischen Gründen, die für die Ausführung nur eine flache Modulstruktur ermöglichen, werden alle Automaten zur Übersetzungszeit in flache Strukturen überführt. In Anlehnung an [AKY99] ergibt sich bei der Überführung zur Laufzeit eine doppelt exponentielle Komplexität. Wie in Abbildung 5.7 werden SHA, die auf maximal 'd' Unterautomaten mit maximal 'n' Lokationen basieren, in Produktautomaten überführt. Im schlechtesten Fall besitzen die Produktautomaten 'n^d' Lokationen, die wieder auf SHAs basieren können. Sei der längste Pfad mit einer Tiefe von 'm' gekennzeichnet, so ergibt sich mit $O(n^{d^m})$ eine doppelte Exponentialität bezüglich der Tiefe ' m '. Somit wird auch für das Problem der Erreichbarkeit in diesem Fall eine Zeit von $O(n^{d^m})$ notwendig. Hierarchisch hybride Automaten, die nicht wohlstrukturiert sind, besitzen im schlechtesten Fall durch das Kopieren aller zur Synchronisation notwendigen Signale, die aus den unteren Ebenen kopiert werden, einen exponentiellen Speicherplatzbedarf. Um dies zu vermeiden, ist für die Ausführung der symbolischen Simulation die Forderung nach der Wohlstrukturiertheit gestellt worden.
Aufgrund des bereits beschriebenen technischen Problems der flachen Modulhierarchie werden HHAs während einer vorherigen Übersetzungsphase in flache Automaten überführt. Wie in [AY01] beschrieben, wird die flache Struktur jedoch gegenüber der modularen Struktur im schlechtesten Fall exponentiell vergrößert, da z.B. Automatentypen aus Bibliotheken auf verschiedenen Hierarchiestufen mehrfach instantiiert werden können. Eine Lösung ist die Vermeidung der Überführung, indem Ansätze aus [ACM06,

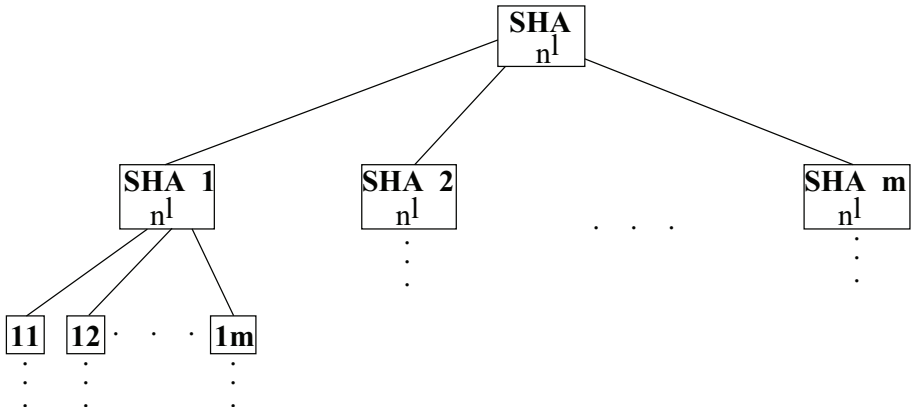


Abbildung 5.6: Nutzung von SHA auf unteren Hierarchieebenen ohne Produktbildung

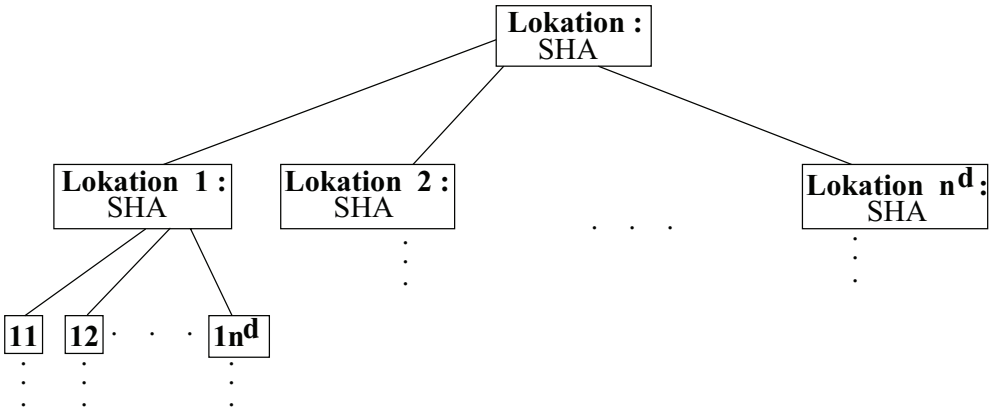


Abbildung 5.7: Nutzung von SHA auf unteren Hierarchieebenen mit Produktbildung

AM06, AAB+07] genutzt werden, wobei Läufe als *geschachtelte Wörter* aufgefasst und entsprechend der Semantik für Läufe aus [AGLS06] abgearbeitet werden können. Um die Komplexität der symbolischen Simulation modularer, hybrider Systeme beruhend auf paralleler und sequentieller Komposition weiterhin einzuschränken, können Ansätze zu Schlussfolgerungstechniken über modularen Strukturen [HK97] wie annahme-garantiertes Schlussfolgern (assume-guarantee reasoning) [HMP01, Rus01, AG04] in unsere Methode integriert werden. Das *annahme-garantierte Schlussfolgern* geht davon aus, Verifikationsanforderungen in Teilanforderungen an einzelne Komponenten des Gesamtsystems zu untergliedern, wobei nicht jede Komponente isoliert betrachtet werden kann, sondern im Zusammenhang mit Annahmen über die Umgebung überprüft werden muss. Für die parallele Komposition zweier Komponenten X_1 und X_2 gilt beim annahme-garantierten Schlussfolgern:

Wenn die Komponente X_1 eine Eigenschaft P_1 unter der Annahme garantiert,

dass die Komponente X_2 eine Eigenschaft P_2 erfüllt und umgekehrt die Komponente X_2 eine Eigenschaft P_2 unter der Annahme garantiert, dass die Komponente X_1 eine Eigenschaft P_1 erfüllt, dann soll die parallele Komposition $X_1||X_2$ die Eigenschaften P_1 und P_2 ohne weitere Bedingungen garantieren.

Spezielle Klassen hybrider Automaten wurden in unterschiedlichen Artikeln auf entscheidbare Probleme untersucht. Dabei werden die Eigenschaften selbst in Automaten transformiert, deren akzeptierte Sprachen durch die Bildung des Durchschnittes mit der akzeptierten Sprache des zu prüfenden Modells Aussagen:

- zur Erreichbarkeit von Zuständen,
- zum Enthaltensein aller zur Eigenschaft gehörigen Worte in der Sprache des zu prüfenden Modells bzw.
- zum Leersein der Menge gemeinsamer Worte der Eigenschaft und des Modells

zulassen. In Tabelle 5.2 sind für folgende Klassen Ergebnisse zum Problem der Erreichbarkeit, des Sprachenthaltenseins und -leerseins angegeben:

Multirate Timed System [ACH+95] ist ein linear hybrides System, dessen Variablen entweder Aussagen, die an den Transitionen nur auf die Werte 0 bzw. 1 gesetzt werden, oder schiefen Uhren, deren Anstieg in jeder Lokation der festgelegten ganzzahligen Konstante k entspricht und an Transitionen entweder auf 0 zurückgesetzt werden bzw. ihren Wert behalten, entsprechen.

Simple Multirate Timed System [ACH+95] ist ein Multirate Timed System, wobei die Form aller Invarianten und Transitionsbedingungen $x \leq k$ bzw. $k \leq x$ für eine Variable x und eine ganzzahlige Konstante k ist.

2-rate Timed System [ACHH93, ACH+95] ist ein 'Multirate Timed System', wobei der Anstieg einer Variablen in unterschiedlichen Lokationen mit 2 verschiedenen, ganzzahligen Konstanten modellierbar ist.

Simple Integrator System [ACHH93, ACH+95] ist ein linear hybrides System, dessen Variablen entweder Aussagen (an Transitionen nur auf Werte 0 oder 1 gesetzt) oder Integratoren (Anstieg in jeder Lokation ist 0 oder 1, an Transitionen entweder auf 0 zurückgesetzt oder behält seinen Wert), sind. Die Form aller Invarianten und Transitionsbedingungen ist $x \leq k$ bzw. $k \leq x$ für eine Variable x und eine ganzzahlige Konstante k.

Rectangular Automaton [Hen96] ist ein hybrider Automat, dessen Aktivitäten unabhängig von den Lokationen und deren Variablen paarweise unabhängig sind. In jeder Lokation ist die erste Ableitung einer jeden Variablen als unendlicher Bereich möglicher Werte gegeben, die sich nicht während der Transitionen ändert. Der Wert jeder Variablen ist mit jeder Transition entweder links unverändert oder verändert

sich nichtdeterministisch innerhalb eines gegebenen Bereiches von möglichen Werten. Das Verhalten der Variablen ist entkoppelt, da die Bereiche möglicher Werte und Ableitungswerte einer Variablen nicht von dem Wert oder dem Ableitungswert einer anderen Variable abhängen kann.

Initialized Rectangular Automaton [HKPV98] ist ein 'Rectangular Automaton', wenn für jede kontinuierliche Variable x und alle aufeinanderfolgenden Lokationen l_i und l_{i+1} gilt, wenn die Aktivität 'act(x)' in l_i ungleich der Aktivität 'act(x)' in l_{i+1} ist, dann ist der Wert von x in der Aktion der Transition von l_i und l_{i+1} geändert worden.

Singular Automaton [Hen96] ist ein 'Rectangular Automaton', wobei die Aktivitäten von der Form $\dot{x} = k$ für eine beliebige Variable x und eine feste Konstante k für alle Lokationen sind.

Multisingular Automaton [Hen96] ist ein 'Rectangular Automaton', wobei die Aktivitäten aller Lokationen von der Form $\dot{x} = k$ für eine beliebige Variable x und eine in jeder Lokation beliebige Konstante k sind.

Triangular Automaton [Hen96] ist ein 'Rectangular Automaton', wobei die Wertebereiche von Variablen nur von links oder nur von rechts begrenzt sein können.

Initialized Multirate Automaton [HKPV98] ist ein 'Initialized Rectangular Automaton', bei dem der Bereich des Anstiegs einer jeden Variable x in jeder Lokation l einelementig ist.

Klasse	*Erreichbarkeit*	*Enthaltensein*	*Leersein*
Simple Multirate Timed System	entscheidbar		
2-rate Timed System	unentscheidbar		unentscheidbar
Simple Integrator System	unentscheidbar		unentscheidbar
Rectangular Automaton		zeit-abstrakte Läufe EXPSPACE-entscheidbar	PSPACE-entscheidbar
Multisingular Automaton	unentscheidbar		
Triangular Automaton	unentscheidbar		
Initialized Multirate Automata	PSPACE-vollständig	PSPACE-vollständig	
		Tabelle 5.2 siehe nächste Seite	

Tabelle 5.2 siehe vorige Seite

Klasse	*Erreichbarkeit*	*Enthaltensein*	*Leersein*
Initialized Rectangular Automata	PSPACE-vollständig	PSPACE-vollständig	

Tabelle 5.2: Entscheidbarkeit von Klassen hybrider Systeme

In [HKPV98] wurden für spezielle 'Rectangular' und 'Triangular' Automaten weitere Untersuchungen bezüglich eingeschränkter Anstiegsraten für Uhren und kontinuierliche Variablen vorgenommen. Weiterhin wurden verschiedene mathematische Theorien und Verfahren wie:

- die Theorie der O-Minimalität im Zusammenhang mit der Bisimulation zur Bestimmung eines endlichen Zustandsquotienten mit äquivalenten Erreichbarkeitseigenschaften zum originalen hybriden System [LPS00, BM05b],
- stückweise affine Abbildungen (piecewise affine maps) [AS02] bzw.
- Pfaffian Funktionen [KV04], welches Polynome über echten analytischen Funktionen sind, wobei die analytischen Funktionen der Erfüllung von Differentialgleichungen dienen,

der Untersuchung von Entscheidbarkeitsfragen für den Bereich der hybriden Systeme zugrunde gelegt. Weitere Ansätze werden zur Verifikation und Erreichbarkeitsanalyse hybrider Systeme mit Sicht auf angenäherte, vereinfachte Systeme, die ein äquivalentes Verhalten aufweisen, genutzt :

- die Reduktion des Verhaltens hybrider Systeme [GHKR98] zu Verhaltensweisen endlicher Zustandsautomaten und
- die Abstraktion hybrider Systeme durch Approximationsverfahren bei der schrittweisen Berechnung von Polyedern über den kontinuierlichen Variablen [ADI06],
- die Abstraktion als 'Over-Approximation' mit rückwirkender Verfeinerung über Gegenbeispiele auf der Basis endlicher Transitionssysteme [CFH$^+$03] und auf der Basis regulärer Sprachen [KRS07], wobei 'Over-Approximation' von folgender Tatsache ausgeht:

  > Gilt eine Eigenschaft in einer Abstraktion, so gilt diese auch im konkreten Modell. Schlägt eine Eigenschaft in der Abstraktion fehl, so wird ein Gegenbeispiel im abstrakten Modell erarbeitet. Wenn dieses Gegenbeispiel auch im konkreten Modell erfolgreich simuliert werden kann, so ist es ein realistisches Beispiel, anderenfalls ist das Gegenbeispiel unecht und führt zu einer rückwirkenden Verfeinerung.

In Transitionssystemen werden logische Aussagen mit Zuständen verknüpft. Bei der Untersuchung aufgrund regulärer Sprachen können solche Informationen an die Transitionen gebunden werden. Auf diese Art und Weise werden Schwierigkeiten während der rückwirkenden Verfeinerung, die in Transitionssystemen durch die Einführung zusätzlicher Zustände bei der Teilung eines abstrakten Zustandes entstehen, vermieden.

- die aufgelockerte (relaxation) Abstraktion auf einer Teilmenge der kontinuierlichen Variablen des originalen hybriden Systems [JKWC07].

In [AMPS08] ist die Frage der Entscheidbarkeit in hybriden Automaten im Zusammenhang mit dem Nachweis von Anfragen nach der Sicherheit noch einmal umfassend untersucht worden, wobei herausgestellt wird, dass viele hybride Systeme im nur 2-dimensionalen Bereich bereits unentscheidbar sind. Diese Tatsache weist auf die Problematik hin, die beim Einsatz hybrider Systeme, welche durch eine Vielfalt an kontinuierlichen Variablen mehrere Dimensionen besitzen, zur Modellierung realer Anwendungen für exakt auszuführende Verifikationen entsteht.
Die Entscheidbarkeit der hybriden Systeme hängt grundsätzlich von den zur Modellierung verwendeten Constraints ab. Zur Überprüfung der Akzeptanz von Zeitwörtern ist für unseren Einsatz somit die Mächtigkeit möglicher Berechnungsdienste des Constraintlösers, welcher einen speziellen Algorithmus zum Lösen erlaubter Constraints eines Constraintsystems im Einklang mit der Constrainttheorie implementiert, von Bedeutung. Entsprechend [FA97] sollte der Constraintlöser folgende Berechnungen durchführen können:

Konsistenztest: Erfüllbarkeit eines Systems vorliegender Constraints in einer Constraintheorie,

Simplifikation: Vereinfachbarkeit von Constraints bis zu Normelformen,

Determination: Bestimmbarkeit des Wertes einer Variablen bis zur kleinsten Normalform,

Projektion: Entfernbarkeit von existentiell quantifizierten Variablen,

Folgerungstest: Entscheidbarkeit von Implikationen zwischen Constraints, die erfolgreich oder gescheitert sein bzw. bis zur Entscheidung verzögert werden kann und

Negation: Umkehrbarkeit von Constraints, die oft unabhängig von den eigentlichen Constraintsystemen durch separat zur Verfügung stehende Constraintsymbole realisiert wird, wodurch jedes negierte Constraint für sich im positiven Kontext behandelt werden kann.

Da in unserem Ansatz mit der Beschreibung von hybriden Systemen eine spezielle Anwendungsdomäne in CLP besteht, können diese Berechnungsdienste auf praktische Anwendungsfälle reduziert werden, wie z.B. Überspringen von Test durch Backjumping oder

die Einteilung in notwendige und hinreichende Bedingungen aus dem praktischen Kontext zum Vereinfachen der Tests durch das Weglassen nicht relevanter Constraints.

Kapitel 6

Sprachbeschreibungen mit MODEL-HS und VYSMO

Nachdem im Kapitel 4 die formalen Grundlagen der textuellen Sprache MODEL-HS und deren zugeordnete, graphische Notation VYSMO eingeführt wurden, werden in diesem Kapitel die konkrete Syntax sowie Unterschiede in der Syntax und Semantik zwischen den beiden Notationen vorgestellt.
Wie bereits erwähnt, sind wesentliche Merkmale von SDL und kommunizierenden Automaten in unseren Sprachen als Ausgangspunkte zur Entwicklung sich über Signale synchronisierender, hybrider Automaten genutzt worden. Die Automaten wurden zur guten Lesbarkeit und Wiederverwendung von Teilsystemen aus Bibliotheken um Hierarchien durch Verfeinerung und Abstraktion im Sinne sequentieller und paralleler Kompositionen erweitert. Der Ansatz des Nachweises von Eigenschaften mit Hilfe der Akzeptanz von Wörtern führt zu einer besonderen Behandlung der Endlokationen bei der Verfeinerung und Abstraktion, wie dies in vorherigen Untersuchungen hybrider Automaten noch nicht erfolgt ist. Im Folgenden werden die Notationen von MODEL-HS und VYSMO bezüglich konkreter syntaktischer Primitiven nebeneinander anhand kleiner Beispiele illustriert. Einzelheiten können der Grammatik von MODEL-HS im Anhang D entnommen werden. Ein größeres Beispiel im nächsten Kapitel verdeutlicht noch einmal Zusammenhänge zwischen den einzelnen Elementen.

6.1 Hybrides System

Ein hybrides System wird als eine Menge von Prozessen aufgefasst, die miteinander und offen mit der Umgebung in ständiger Wechselwirkung stehen. Die Zusammenfassung mehrerer wechselwirkender Prozesse zu einer Einheit wird in MODEL-HS als *Block* und in VYSMO als *synchronisierend hybrider Automat* bezeichnet. Das *System* selbst bildet in diesem Fall einen speziellen Block bzw. synchronisierend hybriden Automaten. Besteht ein System aus genau einem Prozess, so entspricht das Verhalten des Systemblockes in

MODEL-HS semantisch dem Verhalten eines Automaten für diesen Prozess. In VYSMO ändert sich in diesem Fall die Beschreibung, indem das System basierend auf einem Prozess als hierarchisch hybrider Automat notiert wird.
In MODEL-HS wird ein System als ausgezeichneter Block mit einem einführenden Schlüsselwort **system** und abschließenden Schlüsselwort **endsystem** spezifiziert. In VYSMO wird ein System je nach Anzahl der wechselwirkenden Prozesse über einen synchronisierend bzw. hierarchisch hybriden Automaten modelliert. Ein *synchronisierend hybrider Automat* ist in seiner Wirkungsweise einem *Block* aus MODEL-HS und ein *hierarchisch hybrider Automat* einem *Automaten* aus MODEL-HS gleichzusetzen. Eine eigenständige Syntax zur Spezifikation des Systems wurde in VYSMO nicht eingeführt. Im nächsten Abschnitt wird ein System auf der Grundlage eines detaillierten Beispiels beschrieben.

6.2 Blöcke und synchronisierend hybride Automaten

Blöcke und *synchronisierend hybride Automaten* beschreiben abgeschlossene Einheiten in einem System, die in Bibliotheken zusammengefasst und in weiteren Systembeschreibungen wiederverwendet werden können. Mit Hilfe dieser beiden Beschreibungsstrukturen kann von detaillierten Wechselwirkungsbeziehungen zwischen Prozessen abstrahiert werden.
Blöcke in MODEL-HS enthalten Unterblöcke und Automaten, die sich entsprechend auftretender Ereignisse zu bestimmten Zeitpunkten synchronisieren. Synchronisierend hybride Automaten in VYSMO setzen sich aus weiteren synchronisierend hybriden Automaten und hierarchisch hybriden Automaten zusammen, die sich entsprechend auftretender Ereignisse zu bestimmten Zeitpunkten synchronisieren.
Ein Block und ein synchronisierend hybrider Automat werden wie in Abbildung 6.1 spezifiziert.

Name [Formale Parameter]**;**
[Import]
[Deklaration]
[Synchronisation]
endblock Name**;**

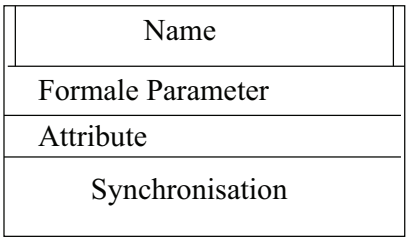


Abbildung 6.1: Block und Synchronisierend hybrider Automat

Blöcke sind in Bibliotheken oder Deklarationsabschnitten übergeordneter Blöcke vordefiniert. Die Beschreibungen der Bibliotheken und Deklarationsabschnitte weisen mit einem Schlüsselwort **blocks** auf die Art der diesem Schlüsselwort folgenden Module hin, so dass die Definition eines Blockes selbst kein einführendes Schlüsselwort benötigt.
Der synchronisierend hybride Automat zeichnet sich durch die seitlichen Doppellinien

im Namensfeld aus, wodurch eine schnelle Unterscheidung zum hierarchisch hybriden Automaten erreicht wird.

6.2.1 Formale Parameter

Sowohl Blöcke als auch synchronisierend hybride Automaten können Parameter und Signale aus der Umgebung empfangen als auch an die Umgebung senden. In der Liste der *formalen Parameter* eines Blockes und eines synchronisierend hybriden Automaten können:

Invarianten, Aktivitäten, Parameter, Uhren, Variablen bzw. Signale

auftreten.

Invarianten und Aktivitäten

Formal deklarierte *Invarianten* und *Aktivitäten* sind Parameter für mögliche Bedingungen und Verlaufsformen, unter denen kontinuierliche Größen des Blockes überwacht und gesteuert werden können. Invarianten werden als Liste, eingeführt durch das Schlüsselwort **invariant** und Aktivitäten als Liste, geführt durch das Schlüsselwort **activity**, angegeben. Die Parameter werden über Namensaufrufe abgearbeitet, was semantisch bedeutet, dass eine textmäßige Ersetzung durch die zugehörig aktuellen Parameter vorgenommen wird. Als einfache Variablen können diese Invarianten und Aktivitäten in jeder beliebigen mathematischen Form ersetzt werden. Als mathematisch vordefinierte Terme und Gleichungen bzw. Ungleichungen müssen die aktuellen Invarianten und Aktivitäten dieser Form textuell entsprechen.

Beispiel 6.2.1
Annahme: G *kontinuierliche Variable,*
X *und* Z *Platzhalter für beliebige Terme aus syntaktischer Sicht*
$G < X$ *ist eine formale Invariante und*
$\dot{G}$: X^Z *ist eine formale Aktivität,*
dann gilt: $S < D+2$ *ist eine gültige aktuelle Invariante und*
$\dot{S}$: 3^S *eine gültige aktuelle Aktivität,*
doch $S > 5$ *entspricht nicht der mathematischen Form*
der formalen Invariante und
$\dot{S}$: $X+Z$ *nicht der mathematischen Form*
der formalen Aktivität,
da die Invariante eine Kleinerbedingung erfüllen
und die Ableitung exponentiell sein muss.

Das Fehlen formaler Invarianten und Aktivitäten wird technisch in MODEL-HS und VYSMO unterschiedlich behandelt. Sind in einem Block keine formalen Invarianten und

Aktivitäten angegeben, so wird standardmäßig die Invariante mit dem Wert 'true' belegt und die Menge der Aktivitäten als leere Menge angenommen. In einem synchronisierend hybriden Automaten dagegen werden ohne weitere Überprüfung und Einschränkung die in dem übergeordneten Automaten definierten aktuellen Invarianten und Aktivitäten zugeordnet.

Parameter, Uhren und Variablen

Größen, die von der Umgebung an Blöcke und synchronisierend hybride Automaten übergeben werden und während einer Ausführung des Prozesses in ihrem Wert nicht änderbar sind, werden als nachstehende Liste von *Parametern* hinter einem Schlüsselwort **parameter** deklariert.
Uhren bezeichnen spezielle Variablen zur Angabe des Taktes während der Abläufe einzelner Prozesse und sind in beiden Sprachen durch das einführende Schlüsselwort **clock** gekennzeichnet.
Die Menge von *Variablen* unterteilt sich in diskrete, kontinuierliche und Steuervariablen. *Diskrete Variablen* werden durch das Schlüsselwort **discrete**, *kontinuierliche Variablen* durch das Schlüsselwort **continuous** und *Steuervariablen* durch das Schlüsselwort **control** angekündigt. Die Steuervariablen werden nach ihrem Gültigkeitsbereich, der einen einzelnen Automaten oder einen ganzen Block umfassen kann, unterschieden. Steuervariablen in Blöcken bzw. synchronisierend hybriden Automaten dienen der Zusammenfassung textueller Strukturen wie z.B. Felder verschiedener Attributwerte oder eine Anzahl gleichartiger Unterblöcke, die an einer Synchronisation beteiligt sind. Diese Variablen werden durch ein Schlüsselwort **in_block** eingeführt. Im später folgenden Beispiel 6.2.3 wird eine Steuervariable in dem praktischen Kontext der Modellierung eines Protokolls verwendet.

Signale

Signale symbolisieren diskret auftretende Ereignisse, die als spezielle Variablen einmal geschrieben und mehrfach gelesen werden können. Diese Signale werden durch das Schlüsselwort **signal** eingeleitet. Signale, die von der Umgebung empfangen werden, sind weiterhin durch ein Schlüsselwort **in** gekennzeichnet und Signale, die an die Umgebung gesendet werden, durch ein Schlüsselwort **out**.

6.2.2 Import

In MODEL-HS können Blöcke und Automaten aus einer Bibliothek eingebunden oder in einem Deklarationsteil definiert werden, wodurch zwischen einem Abschnitt Import und Deklaration zu unterscheiden ist. Der Abschnitt zum *Importieren* wird durch:

import from Bibliotheken

festgelegt und nachfolgende Schlüsselwörter **block_import** bzw. **automaton_import** weisen auf eingebundene Unterböcke bzw. Automaten hin. In VYSMO werden verwendete Automaten standardmäßig aus Bibliotheken eingebunden, weshalb kein separater Bereich zum Importieren geschaffen wurde.

6.2.3 Deklarationsteil und Attribute

Im *Deklarationsteil der Blöcke* von MODEL-HS kann die Bildung von Instanzen auf der Basis bereits vorhandener Teile anderer hybrider Systeme erfolgen. Der Bereich der *Attribute* für synchronisierende hybride Automaten in VYSMO enthält ausschließlich Merkmale in Form von Variablen, Uhren und Signalen.

Deklaration in MODEL-HS

Der Abschnitt Deklaration eines Blockes in MODEL-HS enthält Deklarationsbereiche für:

Unterblöcke, Automaten, Uhren, Variablen und interne Signale.

Der Deklarationsbereich für die Unterblöcke wird durch das Schlüsselwort **blocks** eingeleitet und dient auf der einen Seite der Instanziierung von aus Bibliotheken eingebundenen Blöcken als auch auf der anderen Seite zur Beschreibung neu geschaffener Unterblöcke, die genau einmal genutzt werden und einen Spezialfall darstellen, der mit hoher Wahrscheinlichkeit keine Wiederverwendung findet. Durch die Verwendung derartiger Unterblöcke wird ein unnötiger Verbrauch an Speicherplatz in Bibliotheken vermieden. Die Instanziierung erfolgt durch Einträge folgender Form:

Instanz1 [Anzahl1], Instanz2 [Anzahl2], ... **:** Blockname [Arität].

Der Menge konkreter Instanznamen 'Instanz1', 'Instanz2' usw. folgt ein durch einen Doppelpunkt getrennter Name des aus einer Bibliothek zu instanziierenden Blockes. Soll der Name einer konkreten Instanz für mehrfach auftretende, durch den eingebundenen Block spezifizierte Prozesse ohne Namensänderung verwendet werden, so können die konkreten Instanzen durch einen Index unterschieden werden. Der Index läuft dann von 1 bis zur angebenen 'Anzahl' der zugehörigen Instanz. Wurden verschiedene Unterblöcke mit dem gleichen Blocknamen eingebunden, so kann durch die Angabe der Arität des auszuwählenden Blockes eine eindeutige Beschreibung erstellt werden.
Der Deklarationsbereich für die Automaten wird durch das Schlüsselwort **automata** eingeleitet und dient ähnlich dem Deklarationsbereich für die Unterblöcke der Instanziierung von aus Bibliotheken eingebundenen Automaten und zur Beschreibung neu geschaffener Automaten. Auch hier bilden neu geschaffene Automaten einen Spezialfall und Instanzen werden als Einträge folgender Form in die Spezifikationen aufgenommen:

Instanz1 [Anzahl1], Instanz2 [Anzahl2], ... **:** Automatenname [Arität].

Die Bedeutung dieser Einträge entspricht der Bedeutung der Einträge von Instanzen für die Unterblöcke.
Die Uhren und Variablen werden wie im Abschnitt für die formalen Parameter deklariert. Interne Signale werden wie die Signale der formalen Parameterliste durch das Schlüsselwort **signal** eingeleitet. Diese Signale werden nur zwischen Unterblöcken und Automaten innerhalb des Blockes ausgetauscht und gelangen nicht in die Umgebung des Blockes. Dabei wird nach Signalen unterschieden, die der Block von den Unterblöcken und Automaten empfängt und Signalen, die aus dem Block an die Unterblöcke bzw. Automaten gesendet werden. Auf diese Art und Weise kann der Weg von Signalen innerhalb des Blockes nachvollzogen werden. Die Deklaration und Beschreibung der Wege wird durch unterschiedliche Mechanismen in MODEL-HS und VYSMO verwirklicht. In MODEL-HS wird im Deklarationsteil eine einfache Liste der Signalnamen aufgeführt und erst im Synchronisationsteil eine noch zu klärende Punktnotation genutzt, um den Weg der Signale zu beschreiben. Im Gegensatz zu synchronisierend hybriden Automaten aus VYSMO wird auf diese Art und Weise eine automatische Umbenennung von Signalen im Austausch zwischen Automaten eingesetzt. In SDL wurden Kanäle als Wege für mehrere Signale geschaffen, die anstelle einzelner Signale in den Schnittstellen definiert werden und so eine Anwendung der Spezifikationen für große, komplexe Systeme ermöglichen. Für die Untersuchung des Ansatzes der symbolische Simulation besitzt die Vorgehensweise der expliziten Beschreibung einzelner Signale in den Schnittstellen den Vorteil, die Akzeptanz von Wörtern auf der Basis von Signalen als Repäsentanten der Symbole der Wörter abstrakt und funktional ohne Einblick in die innere Struktur von Blöcken und Automaten überprüfen zu können.

Attribute in VYSMO

Der Abschnitt der *Attribute* in einem synchronisierend hybriden Automaten in VYSMO enthält im Gegensatz zu dem Deklarationsteil eines Blockes keine explizierte Deklaration von Automaten, da diese nur aus Bibliotheken eingebunden werden können und deren Instanziierung im Synchronisationsteil stattfindet. Die Deklaration der Uhren und Variablen wird wie im Deklarationsteil der Blöcke vorgenommen. Interne Signale werden in synchronisierend hybriden Automaten mit dem Schlüsselwort **signal** eingeleitet und durch die Schlüsselworte **send** und **receive** in Signale eingeteilt, die an Unterautomaten gesendet und von Unterautomaten empfangen werden. Diese Einteilung hat den Vorteil, dass bereits im Deklarationsteil offensichtlich wird, auf welchen Wegen Signale ausgetauscht werden.

6.2.4 Synchronisationsbereiche

Die *Synchronisationsbereiche* in Blöcken und synchronisierend hybriden Automaten unterscheiden sich nicht nur durch die Beschreibungsart, sondern auch durch die Art der Verknüpfung von Instanzen zur Synchronisation.

Synchronisation in MODEL-HS

Der Abschnitt zur *Synchronisation in einem Block* wird durch das Schlüsselwort **synchronisation** bekanntgegeben. Dem Schlüsselwort kann eine Anfangsverbindung von der Umgebung zu dem gesamten Block folgen, an der Bedingungen als Wächter und Zuweisungen als Aktionen auftreten, welche für alle Unterblöcke und Automaten zu Beginn eines Laufes gültig sind. Die Anfangsverbindung wird durch das Schlüsselwort **initialisation** eingeführt. Im Synchronisationsabschnitt wird nach dem Schlüsselwort **connection** beschrieben, ob und in welcher Art und Weise sich Automaten und Unterböcke synchronisieren. Dazu wird jede an den Synchronisationen teilnehmende Instanz eines Unterblockes bzw. Automaten in der Form von:

Instanzname **(** Aktuelle Parameter **)**

aufgeführt und durch einen der folgenden Verknüpfungsoperatoren mit den anderen Instanzen verbunden:

|| Parallelität für vollständig unabhängig nebeneinander auszuführende Prozesse,
> Sequenz für vollständig nacheinander auszuführende Prozesse,
v Alternative für die Auswahl eines Prozesses aus einer Menge von Prozessen bzw. zur dynamischen Bindung einer Instanz an unterschiedliche Typen,
| Nebenläufigkeit für nebeneinander auszuführende Prozesse mit ständiger Wechselwirkung.

Wird die Alternative zur dynamischen Bindung verwendet, in der die Zuordnung eines Automaten- oder Blocktyps zu einer Instanz erst zur Laufzeit entsprechend eingegebener Parameter eines Anwenders erfolgt, so ist die Alternative als ausschließliches 'ODER' zu interpretieren.
In der Liste der aktuellen Parameter sind konkrete Werte für Variablen, Uhren und Signale angegeben, die den Unterblöcken und Automaten zur Ausführung eines Laufes übergeben bzw. zwischen den Unterblöcken und Automaten sowie der Umgebung ausgetauscht werden. Um bei der gleichen Bezeichnung von Variablen, Uhren und Signalen die Wege der Werte ohne Umbenennung eindeutig nachvollziehen zu können, wurde eine Punktnotation eingeführt, wobei den jeweiligen Variablen-, Uhren- und Signalbezeichnern der Name des Unterblockes bzw. Automaten durch einen Punkt vorangestellt wird, aus welchem die Variablen-, Uhrenwerte und Signale stammen:

(Unterblock- | Automatenname) **.** (Variablen- | Uhren- | Signalbezeichner).

Beispiel 6.2.2

Annahme: *In einem Block 'Ampel' seien zwei Automaten 'Autoampel' und 'Fußgängerampel' definiert, die je ein Signal 'grün' aussenden. Das Signal 'grün' der 'Fußgängerampel' wird zur Synchronisation von der 'Autoampel' empfangen. Das Signal 'grün' der 'Autoampel' wird zur Synchronisation in die Umgebung gesendet.*

Parameter: *Somit ergeben sich entsprechend der Punktnotation folgende Spezifikationen in den Parameterlisten:*

formale Parameterliste: *Ampel* **(signal out** *Autoampel.grün* **)**,

aktuelle Parameterliste: *Fußgängerampel* **(** *grün* **)**,

Autoampel **(** *Fußgängerampel.grün, grün* **)**

Die Signale werden über Wege ausgetauscht, die von Automaten und Unterblöcken zum übergeordneten Block oder zur Umgebung bzw. umgekehrt vom Block oder aus der Umgebung zu den Automaten und Unterblöcken führen. Neben den genannten Verknüpfungsmöglichkeiten für die Ausführung von Instanzen unterliegen die Wege der Signale Verknüpfungsvarianten, die sich an die Verknüpfungsoperatoren für Ein- und Ausgabekanten aus 'HyCharts' [GS02] anlehnen lassen. Dabei entspricht:

- 'Kopie einer Eingabe auf genau eine Ausgabe' der Variante eines gesendeten Signals, das genau einmal empfangen wird,
- 'Verbindung mehrerer Eingaben zu einer Ausgabe' der Variante mehrerer gesendeter Signale, die gemeinsam ein empfangenes Signal bilden und
- 'Verzweigung einer Eingabe in mehrere Ausgaben' der Variante eines gesendeten Signals, das mehrfach empfangen wird.

Die Verknüpfungen werden hier über die Parameterlisten in den Schnittstellen der Blöcke und Automaten realisiert. Durch die Anordnung der Parameter und die Zuordnung aktueller Parameter zu formalen Parametern lassen sich die genannten Verknüpfungsvarianten beschreiben. Sind die verknüpften Parameter durch Kommata getrennt, so entspricht die Verknüpfung einer UND - Verknüpfung. Sind die verknüpften Parameter durch Semikoli getrennt, so entspricht die Verknüpfung einer ODER - Verknüpfung. Veranschaulichungen erfolgen im Beispiel 6.2.4 und im vollständigen Beispiel des Kapitels E.3.
Im Fall der Synchronisation mehrerer Instanzen auf der Basis eines gemeinsamen Unterblockes bzw. Automaten, die sich nur durch Indizies unterscheiden, kann eine Programmstruktur folgender Form verwendet werden:

for Index Anfang **to** Ende [**step** Schritt]
InstanzName**[**Index**] (** Aktuelle Parameter**)**
end_for

Mit Hilfe der Schleife wird textuell die Synchronisation zwischen Prozessen zusammengefasst, die mehrere Instanzen mit den zugehörigen aktuellen Parameterlisten betreffen, welche sich im Namen nur durch den Index unterscheiden. Der Index ist eine Variable des Typs 'control', die die Zahl der Durchläufe der Schleife zur Vorübersetzung des Blockes, bei welcher die Synchronisationsverbindungen vervielfältigt werden, angibt. Mit dem Schlüsselwort **step** wird die Schrittweite beim Zählen angegeben, die standardmäßig im Fall des Wertes Eins entfallen kann.

Beispiel 6.2.3
Das vereinfachte CSMA/CD Protokoll aus [TR03] mit den Wechselwirkungsbeziehungen der Abbildung 6.2 veranschaulicht noch einmal die Nutzung einer Schleife.

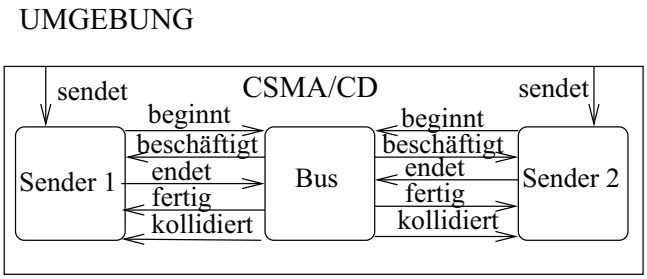


Abbildung 6.2: CSMA/CD Protokoll

Das Protokoll basiert auf den 3 Prozessen 'Sender1', 'Sender2' und dem 'Bus', welche sich über die Signale 'sendet', 'beginnt', beschäftigt', 'endet', 'fertig' und 'kollidiert' synchronisieren. Durch das Signal 'sendet' löst die Umgebung den Sendevorgang in einem Sender aus. Befindet sich der 'Bus' in einem Ruhezustand, so kann der Sender durch das an den Bus gesendete Signal 'beginnt' den Sendevorgang starten. Der Vorgang ist im erfolgreichen Fall nach genau einer Zeit bezeichnet mit 'λ' abgeschlossen. Nach der Zeit 'λ' sendet der Sender ein 'ende' - Signal an den Bus, wobei der Bus das Signal mit 'fertig' bestätigt. Standardmäßig benötigt der Bus nach dem Empfang des Signals 'beginnt' eine Zeit vom Wert 'σ', um ein Signal 'beschäftigt' an alle Sender zur Information auszusenden. Vor dem Ablauf der Zeit 'σ' kann es zum Sendeversuch eines anderen Senders kommen, so dass eine Kollision entsteht, die durch den Prozess 'Bus' mit dem Signal 'kollidiert' angezeigt wird. Das hybride System 'CSMA_CD' ist mit folgenden formalen Parametern ausgestattet:

CSMA_CD **(parameter** *sigma,lambda* **;**
signal in *sendet[1..2]* **)** .

Das Signal 'send' kann mit einem Index aus dem Bereichsintervall '1..2' verbunden werden. Weiterhin werden zwei Automatentypen 'SENDER' und BUS' mit folgenden formalen Parametern importiert:

SENDER **(parameter** *sigma,lambda* **;**
signal in *sendet,beschäftigt,fertig,kollidiert* **;**
out *beginnt,endet* **) ;**
BUS **(parameter** *sigma* **;**
signal in *beginnt,endet* **;**
out *beschäftigt,fertig,kollidiert* **) .**

Es existieren zwei Instanzen von 'SENDER' und eine Instanz von 'BUS':

Sender[2] : SENDER **;**
Bus : BUS .

Da die beiden Sender Signale in der gleichen Art und Weise austauschen, kann die Synchronisation textuell in einer Schleife zusammengefasst werden:

for *i: 1* **to** *2*
Sender[i] **(** *sigma,lambda,sendet[i],beschäftigt,fertig,kollidiert,beginnt,endet* **) ;**
Bus **(** *sigma,Sender[i].beginnt,Sender[i].endet,beschäftigt,fertig,kollidiert* **) ;**
end_for .

Die Schrittweite beträgt in diesem Fall Eins. Die Variable 'i' ist eine Variable vom Typ **control in_block**.

Synchronisation in VYSMO

In VYSMO ist der *Synchronisationsteil* eines *synchronisierend hybriden Automaten* durch Instanzen von synchronisierend und hierarchisch hybriden Automaten und durch zugehörige Verknüpfungsoperatoren gekennzeichnet. Dabei werden die Instanzen wie in Abbildung 6.3 visualisiert.

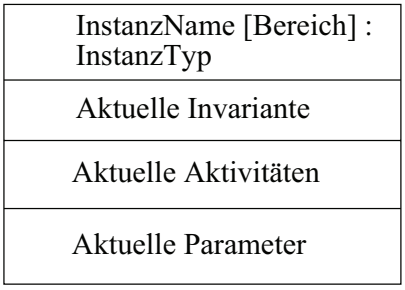


Abbildung 6.3: Instanzen von synchronisierend und hierarchisch hybriden Automaten

Die Typen der Instanzen liegen als graphische Beschreibungen in Standardbibliotheken

vor. Der Name einer Instanz kann mit einer Bereichsangabe über die Anzahl bestehender Instanzen verbunden werden. Die aktuelle Invariante wird als Bedingung über Variablen und Uhren auf der Grundlage der mathematischen Logik formuliert. Aktuelle Aktivitäten stellen Differentialgleichungen über der Menge der kontinuierlichen Variablen und Uhren dar. Zu den aktuellen Parametern zählen alle konkreten Werte, die für formal deklarierte Variablen, Uhren und Signale übergeben bzw. zurückgegeben werden.
Die Verknüpfungsoperatoren zur Art der Ausführung der Instanzen lassen sich einteilen in:

|| Parallelität für vollständig unabhängig nebeneinander ablaufende Instanzen,

v Alternative für die Auswahl einer Instanz aus einer Menge von Instanzen bzw. zur dynamischen Bindung einer Instanz an unterschiedliche Typen
und

Synchronisationsverbindungen, die eingeteilt werden in:

1. Anfangsverbindungen von der Umgebung zum gesamten Synchronisationsteil mit Gültigkeit für alle aufgeführten Instanzen,

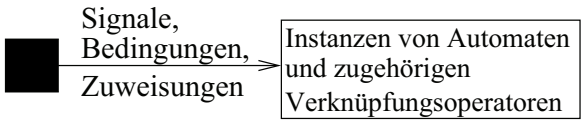


Abbildung 6.4: Anfangsverbindung

wobei die Signale aus der Umgebung zu empfangende Signale darstellen und die Bedingungen und Zuweisungen über und für alle Parameter des Automaten gelten.

2. in Verbindungen von Instanzen zur Umgebung und zurück

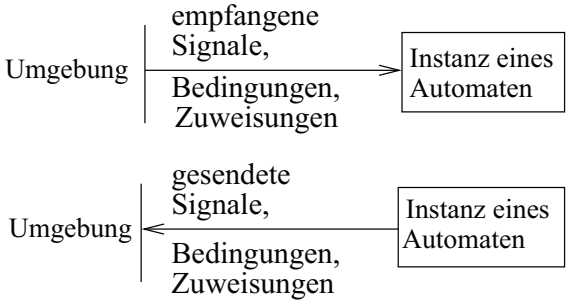


Abbildung 6.5: Verbindung zur und von der Umgebung

sowie

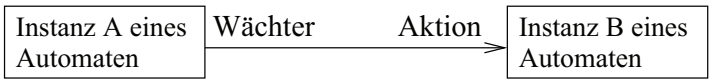


Abbildung 6.6: Verbindung zwischen Instanzen

3. in Verbindungen zwischen Instanzen
   wobei der Wächter Signale enthält, die der übergeordnete Automat von der Instanz A unter angegebenen Bedingungen des Wächters empfangen hat und die Aktion enthält die zugehörig umbenannten Signale, die mit auszuführenden Variablenzuweisungen dieser Aktion an die Instanz B gesendet werden. Sind keine Signale in der Aktion enthalten, so werden die Signale des Wächters ohne Umbenennung zur Synchroniation verwendet. Im Gegensatz zu der verwendeten Punktnotation von MODEL-HS kann die Umbenennung der Signale hier praxisorientiert und nutzerfreundlich erfolgen, da die Signalwege durch die graphischen Verbindungen im Synchronisationsteil festgelegt sind.

Die Sequenz und die Nebenläufigkeit bilden Spezialfälle der Synchronisationsverbindungen. Eine Sequenz ist die Verbindung, die durch die Synchronisation über ein Signal der sendenden Instanz entsteht, welches aus einer Transition zu einer Endlokation im Automatentyp dieser sendenden Instanz hervorgeht. Die Nebenläufigkeit ergibt sich aus einer Menge von Synchronisationsverbindungen zwischen Instanzen von Automaten. Neben den Synchronisationsverbindungen bestehen weitere Verknüpfungsvarianten für die Wege der Signale aus und in einen synchronisierend hybriden Automaten, die wie in MODEL-HS über die Anordnung, Zusammenfassung und Trennung von Signalen in den Parameterlisten der Schnittstellen beschrieben werden.

Beispiel 6.2.4
Durch einen stark vereinfachten Ausschnitt aus einem Medizinstudium, wie in [TLR06] modelliert, sollen Verknüpfungsmöglichkeiten in synchronisierend hybriden Automaten beispielhaft veranschaulicht werden. Diese Illustration ist gleichzeitig maßgebend für Verknüpfungsvarianten innerhalb von Blockbeschreibungen in MODEL-HS.
Nach dem Absolvieren des Physikums sind in einem 2. Abschnitt Grundfächer erfolgreich abzulegen, wobei z.B. die 'Allgemeinmedizin' unabhängig von anderen Prüfungen zu bestehen ist und Fächer wie 'Augenheilkunde' und 'Dermatologie' neben Komplexprüfungen über weitere Fachgebiete in fachübergreifenden Leistungsnachweisen abzulegen sind. Dabei kann sich der Student zwischen zwei fachübergreifenden Leistungsnachweisen 'FLN_1' und 'FLN_2' alternativ entscheiden. Zusätzlich zu den Grundfächern ist eine Prüfung in einem Wahlfach zu absolvieren, welches entweder die 'Sportmedizin', die 'Zahnmedizin' oder vertiefend eines der Grundfächer betreffen kann. Erst wenn die Grundfächer und das Wahlfach bestanden wurden, kann ein praktisches Jahr und danach die abschließenden schriftlichen und mündlichen Prüfungen absolviert werden.
Der genannte Ablauf ist im Synchronisationsteil des synchronisierend hybriden Automaten 'Studienverlauf' in 6.7 dargestellt. Parameter wie 'WF' bzw. 'FLN' weisen dabei auf

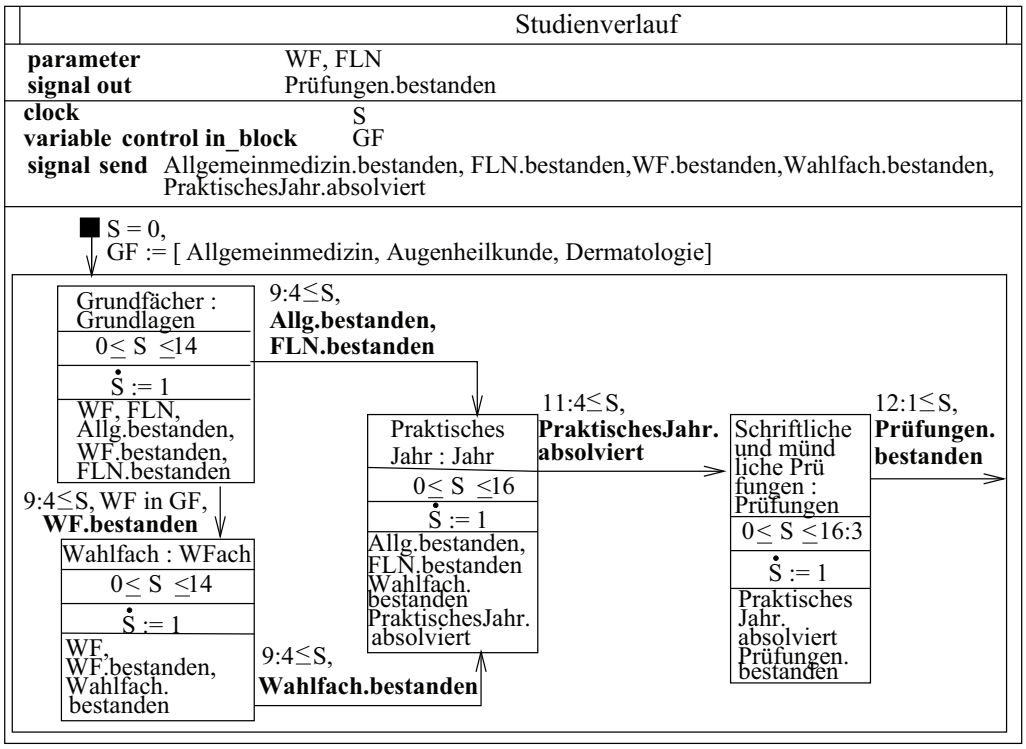


Abbildung 6.7: Vereinfachter Studienverlauf eines 2. Abschnittes des Medizinstudiums

das von einem Studenten ausgesuchte Wahlfach bzw. den gewählten fachübergreifenden Leistungsnachweis hin. Die Systemuhr 'S', die zu Beginn eines jeden Laufes dem Betrachtungszeitpunkt Null entsprechen soll, gibt in Invarianten die zeitlichen Grenzen zur Ausführung eines Prozesses und an den Verbindungswegen die zeitlichen Bedingungen zu einer möglichen Synchronisation durch die an den Verbindungen stehenden Signale an. Die Steuervariable 'GF' für Blockbeschreibungen wird mit einer Liste von abzulegenden Grundfächern initialisiert. Bis auf das Signal 'Prüfungen.bestanden' werden alle weiteren Signale zur Synchronisation innerhalb des Automaten 'Studienverlauf' verwendet. Den hauptsächlichen Ablauf betreffend werden alle Teilprozesse nebenläufig ausgeführt. Die nebenläufige Ausführung wird durch Zeitbedingungen gesteuert, die in Anzahl an Semestern und Monaten spezifiziert werden. Die Zahl der Semester sind dabei vor dem Doppelpunkt und die Zahl der Monate nach dem Doppelpunkt aufgeführt. Das Fach 'Allgemeinmedizin' kann dementsprechend nach 9 Semestern und 4 Monaten abgeschlossen werden, muss jedoch bis zum Ende des 14. Semesters bestanden worden sein. Streng sequentielle Abfolgen bestehen, wenn die gesendeten Signale zu einer Endlokation in einem Automaten führen. Zum Beispiel zwischen dem Ablegen des Wahlfaches und dem Praktischen Jahr ist die Abfolge streng sequentiell, da das Signal 'Wahlfach.bestanden' erst ausgesendet wird, wenn eine als erfolgreich bezeichnete Endlokation im Automatentyp 'WFach' erreicht wurde.

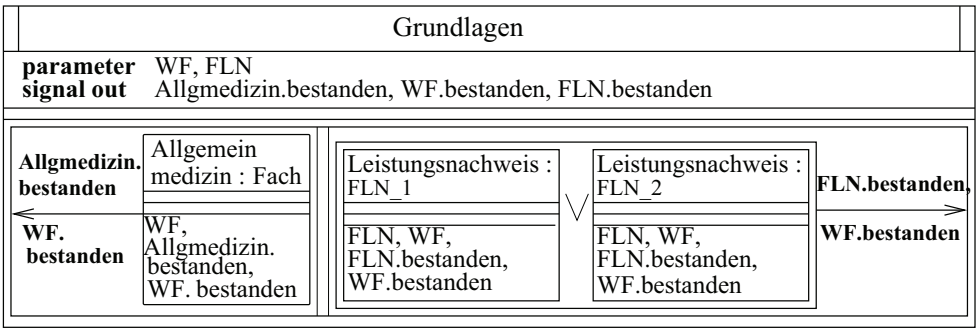


Abbildung 6.8: Grundlagenfächer

Im synchronisierend hybriden Automaten 'Grundlagen' der Abbildung 6.8 sind die Prüfungsabläufe der 'Allgemeinmedizin' und des ausgewählten fachübergreifenden Leistungsnachweises parallel unabhängig voneinander auszuführen. Der fachübergreifende Leistungsnachweis kann je nach Wahl entweder über den Ablauf eines Automatentypen 'FLN_1' oder 'FLN_2' ausgeführt werden. Hier liegt eine dynamische Bindung der Instanz 'Leistungsnachweis' an einen der Typen FLN_1' oder 'FLN_2' bezüglich der Belegung des Parameters 'FLN' vor. Vereinfachend, jedoch auf den ersten Blick aufgrund der textuellen Notation nicht sofort erkennbar, kann die dynamische Bindung auch durch eine Instanz mit dem Namen und zugehörigem Typ 'Leistungsnachweis: FLN_1 ∨ FLN_2' angegeben werden, welche durch die graphische Darstellung eine schnellere Übersicht ermöglicht.

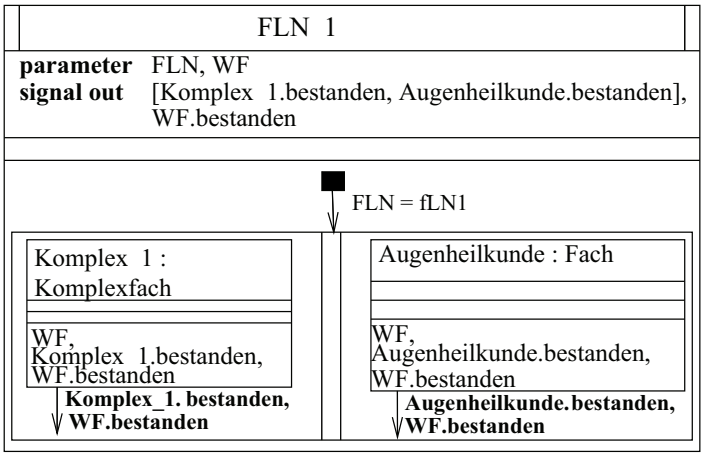


Abbildung 6.9: Fachübergreifender Leistungsnachweis

Die Abbildung 6.9 weist mit dem Automaten 'FLN_1' eine mögliche Verknüpfung für Automaten und eine mögliche Verknüpfung für Verbindungswege auf. In 'FLN_1' können die Prozesse im Synchronisationsteil nur abgearbeitet werden, wenn der Parameter

'FLN' mit dem Wert 'fLN1' bei der Wahl des fachübergreifenden Leistungsnachweises von einem Studenten belegt wurde. Die Ausführung des Prüfungsablaufes für das Fach 'Augenheilkunde' erfolgt vollkommen unabhängig von der Ausführung des Prüfungsablaufes für das Komplexfach 'Komplex_1'. Deshalb sind beide Automaten mit dem Zeichen der parallelen Ausführung verbunden, welches einem Doppelstrich entspricht. Die Ergebnisse der beiden Abläufe, die im erfolgreichen Fall durch die Signale 'Komplex_1.bestanden' und 'Augenheilkunde.bestanden' symbolisiert werden, bilden jedoch gemeinsam den erfolgreichen Abschluss des fachübergreifenden Leistungsnachweises, der in Abbildung 6.8 durch das Signal 'FLN.bestanden' symbolisiert wurde. Die UND-Verknüpfung, bei welcher mehrere gesendete Signale als ein Signal empfangen werden, wird durch die Zusammenfassung der Signale 'Komplex_1.bestanden' und 'Augenheilkunde.bestanden' mit eckigen Klammern in der Parameterliste der Schnittstelle des synchronisierend hybriden Automaten 'FLN_1' realisiert.

6.2.5 Beispielsystem in MODEL-HS und VYSMO

Dieser Abschnitt dient der Gegenüberstellung von System- und Blockbeschreibungen in MODEL-HS mit der Beschreibung von Systemen und synchronisierend hybriden Automaten in VYSMO. Anhand eines kleinen Beispiels, welches im Abschnitt 6.3.5 vervollständigt wird, werden allgemein formulierte Beschreibungsmittel vertiefend dargestellt.

Beispiel 6.2.5
Das System zur Steuerung des Wasserstandes in einem Wasserbehälter wird durch zwei Prozesse, der Füllstandsanzeige und einer Pumpe, realisiert. Die Abläufe der Prozesse werden auf der Basis der hierarchisch hybriden Automatentypen 'Wasserstand' und 'Schalter' spezifiziert und im Abschnitt 6.3.5 detailliert dargestellt. Die Spezifikation in MODEL-HS wird wie folgt erstellt:

```
system Wasserbehälter (parameter PumpLeist1, PumpLeist2;
                       signal in   beginn);

import from Library
    automaton_import
        Wasserstand(invariant WasserInv;
                    activity WasserAct;
                    parameter Leist1, Leist2;
                    signal synchron in   wasser_an, schneller
                                   out   pump_an
                           refine  out   pump_aus);
        Schalter(invariant SchaltInv;
                 activity SchaltAct;
                 parameter Leist;
                 signal synchron in   an
```

out *voll_L*
refine **in** *aus);*

declaration
automata
Anzeige: Wasserstand;
Pumpe: Schalter;
clock *S;*
signal *pump_an, pump_aus, voll_Leist;*

synchronisation
initialisation
S : 0;
connection
Anzeige(0 ≤ S, {S: S+1}, PumpLeist1, PumpLeist2, beginn, voll_Leist(S: 0),
pump_an(10 ≤ S), pump_aus(10 ≤ S)) |
Pumpe(0 ≤ S, {S: S+1}, PumpLeist1, pump_an(S: 0), voll_Leist(4 ≤ S),
pump_aus(S: 0))
endsystem *Wasserbehälter.*

Demgegenüber wurde das Systems mit graphischen Elementen von VYSMO in der Abbildung 6.10 beschrieben. Der Wasserstand kann über zwei Leistungsstufen reguliert wer-

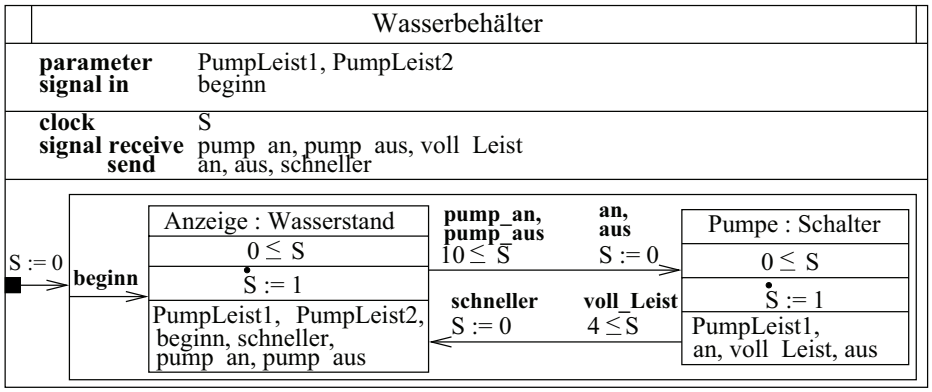


Abbildung 6.10: Steuerung des Wasserstandes

den, die in den Parametern 'PumpLeist1' und 'PumpLeist2' festgelegt sind. Ein Signal 'beginn' wird in der Umgebung ausglöst, um den Prozess der Steuerung zu starten. Diese drei Argumente bilden die Schnittstelle des Systems 'Wasserbehälter'. Der 'Wasserbehälter' ist mit einer lokalen Systemuhr 'S' verbunden, die den Takt für den zeitlichen Ablauf des gesamten Prozesses vorgibt. Die Signale 'pump_an', 'pump_aus', 'voll_Leist', 'an', 'aus' und 'schneller' dienen der Synchronisation der beiden Prozesse 'Anzeige' und

'Pumpe'. Diese Signale haben keinen Bezug zur äußeren Umwelt und sind somit lokal deklariert. In der MODEL-HS Beschreibung entfallen die Signale 'an', 'aus' und 'schneller', da die Umbenennung der Punktnotation unterliegt. In diesem Beispiel ist diese Notation aufgrund der Eindeutigkeit der Signalwege nicht notwendig.
Für die symbolische Simulation wird eine Instanz 'Anzeige' vom Automatentyp 'Wasserstand' und eine Instanz 'Pumpe' vom Automatentyp 'Schalter' erzeugt. Die Invarianten der Instanzen '0 $\leq$ S' besagen, in welchem Zeitraum die Prozesse ablaufen dürfen und die Aktivitäten '$\dot{S}$: 1' geben den Takt des Fortschreitens der Simulationszeit während des Ablaufes der Prozesse an. Der Wasserstand wird über die Parameter 'PumpLeist1' und 'PumpLeist2' geregelt. Der Schalter richtet sich nach dem Parameter 'PumpLeist1'. Ob und wann die Pumpe ein- bzw. auszuschalten ist, wird durch den Wasserstand über 'pump_an' und 'pump_aus' gesteuert. Wie schnell sich der Wasserstand verändert, hängt von der Einstellung der Pumpleistung ab, die mit 'voll_Leist' die höchste Leistungsstufe erreicht.

6.3 Automaten und hierarchisch hybride Automaten

Automaten in MODEL-HS und *hierarchisch hybride Automaten in VYSMO* bilden das Verhalten grundlegender Prozesse eines Systems ab. Automaten als Basiselemente zum Aufbau von Blockhierarchien in MODEL-HS und synchronisierend hybriden Automaten in VYSMO werden wie in der Abbildung 6.11 beschrieben.

Name [Parameter]**;**
[Verfeinerung]
Deklaration
Verhalten
endautomaton <Name>**;**

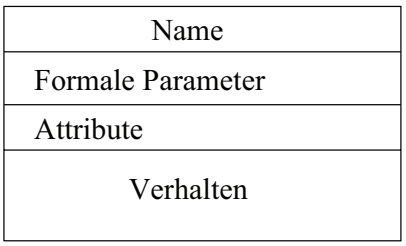


Abbildung 6.11: Automat und Hierarchisch hybrider Automat

Auch Automaten sind in Bibliotheken oder Deklarationsabschnitten übergeordneter Blöcke vordefiniert. Die Beschreibungen der Bibliotheken und Deklarationsabschnitte weisen mit einem Schlüsselwort **automata** auf die Art der diesem Schlüsselwort folgenden Module hin. Der hierarchisch hybride Automat besitzt eine einfache Umrandung für das Namensfeld.

6.3.1 Formale Parameter

Automaten und hierarchisch hybride Automaten tauschen wie Blöcke und synchronisierend hybride Automaten Parameter und Signale mit der Umgebung aus. In der Liste der *formalen Parameter* von Automaten und hierarchisch hybriden Automaten sind enthalten:

Invarianten, Aktivitäten, Parameter, Uhren, Variablen bzw. Signale.

Unterschiede in der Deklaration zu formalen Parametern von Blöcken und synchronisierend hybriden Automaten treten bei den Variablen und Signalen auf.

Variablen

Die *Variablen* lassen sich wie für Blöcke und synchronisierend hybride Automaten in diskrete, kontinuierliche und Steuervariablen einteilen. Diskrete und kontinuierliche Variablen stimmen in der Bezeichnung und Bedeutung mit den diskreten und kontinuierlichen Variablen der Blöcke und synchronisierend hybriden Automaten überein. Steuervariablen, die durch das Schlüsselwort **control** angekündigt werden, erhalten als Zusatzbezeichnung das Schlüsselwort **in_automaton**. Hierdurch wird der Gültigkeitsbereich der Steuervariablen auf einen Automaten festgelegt, in welchem die Variable zur Einsparung von Transitionen und Lokationen verwendet wird [Tet00]. Diese Wirkungsweise unterscheidet die Steuervariable von einer als diskret deklarierten Variable.

Beispiel 6.3.1
In den Abbildungen 6.12 und 6.13 ist das Verhalten eines Senders aus dem Beispiel 6.2.3 mit der Steuervariable 'B_B' und ohne Steuervariable dargestellt. Der Schwerpunkt liegt auf der Bedeutung der Steuervariable, so dass das Verhalten des Buses in der Abbildung 6.14 lediglich zur Vervollständigung angegeben ist.

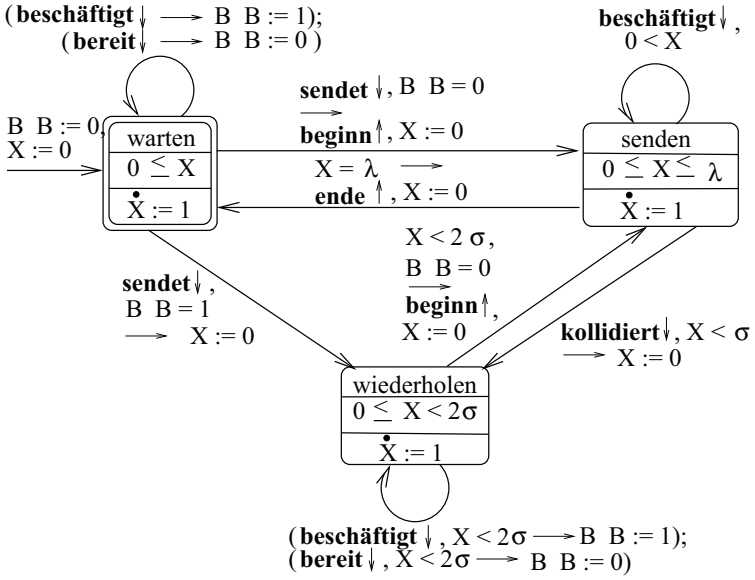


Abbildung 6.12: Sender mit Steuervariable B_B

Ein Sender hat zu Beginn auf die Anforderung der Umgebung zum Versenden einer Nach-

richt zu warten. Erhält der Sender die Anforderung während der Bus keinen Auftrag abzuarbeiten hat, so beginnt der Sendevorgang und führt zur Lokation 'senden'. Wird der Sendevorgang nach 'λ' Zeiteinheiten erfolgreich beendet, so kehrt der Sender mit einem Signal 'ende' zur Lokation 'warten' zurück. Da die Busverbindung mindestens eine Zeit von 'σ' Zeiteinheiten benötigt, um das Signal 'beschäftigt' immer wieder periodisch auszulösen, kann der Bus während des laufenden Sendevorgangs einen weiteren Auftrag von einem anderen Sender erhalten. In diesem Fall wird der Sendevorgang durch das Kollisionssignal 'kollidiert' abgebrochen. Innerhalb einer Zeit von '2σ' Zeiteinheiten kann der Sender den Sendeversuch wiederholen und von der Lokation 'wiederholen' zur Lokation 'senden' zurückkehren. Dasselbe gilt in der Lokation 'warten'. Ist das Bussystem beschäftigt, so kann der Sender den Sendeversuch nach einer Anforderung 'sendet' in der Lokation 'wiederholen' neu starten.
Die Variable 'B_B' speichert die Information über den derzeitigen Zustand des Buses. Im Fall der Belegung von 'B_B' mit dem Wert Null kann der Sendevorgang begonnen werden. Ist der Bus beschäftigt, so muss der Sender nach einer Anforderung der Umgebung zum Versenden einer Nachricht in der Lokation 'wiederholen' auf die Bereitschaft des Buses warten. Sobald ein Sendevorgang erfolgreich abgschlossen wird, sendet der Bus ein Signal 'bereit' aus und für die Sender wird die Variable 'B_B' wieder auf den Wert Null gesetzt.

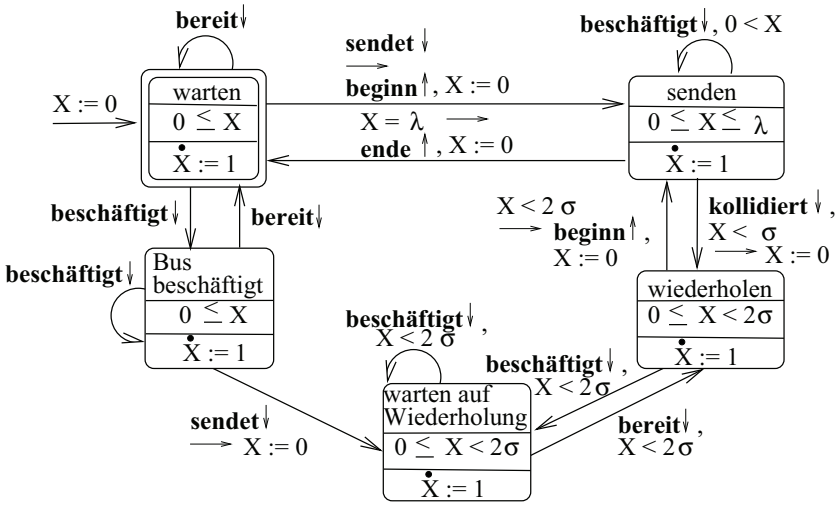


Abbildung 6.13: Sender ohne Steuervariable

Entfällt die Nutzung der Steuervariable 'B_B' wie im Fall der Abbildung 6.13, so werden zwei zusätzliche Lokationen 'Bus beschäftigt' und 'warten auf Wiederholung' sowie neue Übergänge zu und von den neuen Lokationen benötigt. Das Verändern und Überprüfen des Wertes der Steuervariable 'B_B' an den Übergängen der Lokation 'warten' in der Abbildung 6.12 spart die Lokation 'Bus beschäftigt' zusammen mit den verbundenen Transitionen, die mit den Signalen 'beschäftigt' und 'bereit' gekennzeichnet sind,

ein. Die an den Transitionen der Lokation 'wiederholen' durchgeführten Veränderungen und Überprüfungen des Wertes der Steuervariable 'B_B' spart die Lokation 'warten auf Wiederholung' zusammen mit allen verbundenen Transitionen, die mit den Signalen 'beschäftigt' und 'bereit' gekennzeichnet sind, ein.

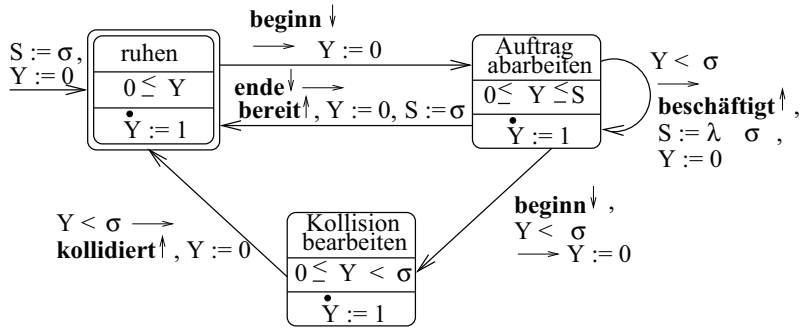


Abbildung 6.14: Verhalten des Buses

Signale

Signale eines Automaten der Sprache MODEL-HS bzw. hierarchisch hybriden Automaten der Sprache VYSMO sind immer in der Schnittstelle des Automaten deklariert, da Signale der Synchronisation mit anderen Automaten und Blöcken dienen. Die Einschränkung des Gültigkeitsbereiches durch lokale Deklaration von in Automaten erzeugten Signalen als Verdeckungsprinzip ist für die technische Sicht der Synchronisation nicht sinnvoll. Diese technische Sichtweise steht im Gegensatz zur möglichen Verdeckung von formal zu untersuchenden Teilwörtern, die aus einer Folge von Signalen in Verbindung mit deren zeitlichem Auftreten symbolisiert werden. Deshalb müssen hier während der symbolischen Simulation der hierarchisch hybriden Automaten abweichende Betrachtungen zu dem Bereich formaler Sprachen vorgenommen werden.
Die deklarierten Signale werden in *Signale zur Synchronisation* und *Signale zur Verfeinerung* eingeteilt. Alle Signale, die mit Übergängen verbunden sind, die nicht zu Endlokationen führen, werden als Signale zur Synchronisation hinter dem Schlüsselwort **synchron** aufgeführt. Diejenigen Signale, welche mit zu Endlokationen führenden Übergängen verbunden sind, werden als Signale zur Verfeinerung hinter dem Schlüsselwort **refine** aufgeführt. Jede der beiden Kategorien wird mit den Schlüsselwörtern **in** und **out** weiterhin in empfangene und ausgesendete Signale untergliedert.

6.3.2 Verfeinerung

Zur *Verfeinerung* sind in MODEL-HS durch:

refinement from Bibliotheken **by** Module

diejenigen Automaten und Blöcke mit deren zugehörigen Bibliotheken anzugeben, auf denen die Verfeinerung einer komplexen Lokation basiert. Dabei werden für die Automaten und Blöcke im Abschnitt 'Module' die Schnittstellen mit den Bezeichnern und den Listen der formalen Parameter spezifiziert. In VYSMO ist vorerst kein separater Bereich zum Verfeinern geschaffen, da die verwendeten Automaten standardmäßig aus vorgeschriebenen Bibliotheken eingebunden werden.

6.3.3 Deklaration und Attribute

Der *Deklarationsteil* eines *Automaten in MODEL-HS* und der *Attributteil* eines *hierarchisch hybriden Automaten in VYSMO* beinhalten lokale Uhren und Variablen. Die Uhren und Variablen werden wie in der Liste der formalen Parameter beschrieben. In MODEL-HS werden zusätzlich alle Lokationsbezeichner eines Automaten hinter dem Schlüsselwort **location** deklariert. Dabei erfolgt eine Unterteilung in einfache und komplexe Lokationen. Komplexe Lokationen können durch Automaten und Blöcke verfeinert werden. Als Deklaration einfacher Lokationen werden deren Bezeichner hinter einem Schlüsselwort **simple** aufgeführt. Die Deklaration komplexer Lokationen erfolgt hinter dem Schlüsselwort **complex** durch die Instanziierung von zur Verfeinerung eingebundener Automaten und Blöcke aus den Bibliotheken bzw. direkt im Deklarationsteil definierten Automaten und Blöcken, die nur einmal verwendet werden. In VYSMO werden Lokationen nicht explizit deklariert, da die Lokationen durch die graphische Gestaltung im Abschnitt der Beschreibung des Verhaltens schnell in der Art, Anordnung und Anzahl erfassbar sind.

6.3.4 Verhaltensbeschreibungen

Die *Verhaltensbeschreibungen* der Automaten und hierarchisch hybriden Automaten in MODEL-HS und VYSMO lassen sich leicht aufeinander abbilden. Die Beschreibung von MODEL-HS wurde hierbei stark an die graphische Notation von VYSMO angelehnt. Ein kleiner Unterschied besteht in der Zuordnung aktueller Parameter zu den formalen Parametern der zur Verfeinerung verwendeten Automaten und Blöcken, die in MODEL-HS in einem getrennten Bestandteil der Beschreibung und in VYSMO in direkter Verbindung mit der komplexen Lokation vorgenommen wird.

Verhalten in MODEL-HS

Die Verhaltensbeschreibung in MODEL-HS wird durch das Schlüsselwort **behaviour** eingeleitet. Im Anschluss an dieses Schlüsselwort erfolgt die Spezifikation des Verhaltens in Abfolge der Bestandteile:

[Parameterzuordnung]
Initialisierung
Terminierung
Transitionen.

Die Parameterzuordnung kann entsprechend vorhandener komplexer Lokationen optional eingefügt werden, was durch die eckige Klammerung zum Ausdruck kommt. Der Bestandteil wird durch das Schlüsselwort **matching** eingeführt. Nach diesem Schlüsselwort werden alle Instanzen und deklarierte Automaten bzw. Blöcke komplexer Lokationen in Verbindung mit der Liste aktueller Parameter aufgeführt, die ein durch Verfeinerung detaillierteres Verhalten aufweisen. Dabei kann wie im Synchronisationsteil der Blöcke die Punktnotation zur Unterscheidung von gleichen Parameterbezeichnern und Richtungsbestimmung von Signalen eingesetzt werden.
Der Initialisierungsteil wird mit dem Schlüsselwort **initialisation** gekennzeichnet. Hier wird mit folgender Syntax:

if Alternative von (Wächter [**->** Aktion])
location Anfangslokation**;**

beschrieben, unter welchen Anfangsbedingungen und mit welchen Zuweisungen welche Lokation zu Beginn eines Prozesses aktiv wird. Dabei können mehrere Wächter mit zugehörigen Aktionen aufgeführt werden, von denen wenigstens eine Verbindung aus Wächter und Aktion Gültigkeit besitzt. Im Wächter sind die zu empfangenden Signale und Anfangsbedingungen für feststehende Parameter enthalten und in der Aktion auszusendende Signale und Zuweisungen als Initialisierung von Uhren und Variablen. Die Aktion kann optional beschrieben werden.
Im Teil der Terminierung werden nach einem Schlüsselwort **termination** alle Bezeichner von Lokationen aufgeführt, die Endlokationen darstellen.
Die Transitionen werden auf das Schlüsselwort **transition** folgend in kontinuierliche Transitionen zur Spezifikation des kontinuierlichen Verhaltens der Lokationen und in diskrete Transitionen zur Spezifikation der diskreten Zustandsänderungen an den Übergängen eingeteilt.
Kontinuierliche Transitionen nach dem Schlüsselwort **continuous** werden mit Hilfe folgender syntaktischer Struktur formuliert:

[**while** Gemeinsame Invariante]
Liste von Tupel mit
Tupel (Gemeinsame Aktivitäten,
Liste Kombinierter Lokationen).

Die 'Gemeinsame Invariante' gilt für alle Lokationen, die im Kontext dieser 'while' Struktur aufgeführt sind. Besitzen die Lokationen keine Teilbedingung in den Invarianten, die als 'Gemeinsame Invariante' ausgeklammert werden kann, so entfällt dieser Teil der

Syntax. Mehrere Lokationen können gleiche Aktivitäten zur Beschreibung der kontinuierlichen Flüsse von Uhren und Variablen besitzen, die explizit in 'Gemeinsame Aktivitäten' ausgeklammert werden. Derartige Aktivitäten werden mit dem Schlüsselwort **do** eingeführt. Ist die Liste der gemeinsamen Aktivitäten leer, so entfällt dieser Bestandteil. Die Liste 'Kombinierte Lokationen' wird mit dem Schlüsselwort **in** an die vorhergehenden Aktivitäten angeschlossen. Eine kombinierte Lokation wird wie folgt spezifiziert:

Bezeichner **(** [**while** Invariante], [**do** Aktivitäten] **)** ,

wobei der Bezeichner den Namen der Lokation, 'Invariante' die Bedingungen und 'Aktivitäten' die kontinuierlichen Flüsse, die sich nur auf die bezeichnete Lokation beziehen, darstellen. Auf diese Art und Weise wurde eine kompakte syntaktische Struktur zur Beschreibung der Abläufe in Lokationen geschaffen, die eine äquivalente Aussagekraft zu streng getrennten Beschreibungen einzelner Lokationen besitzt.
Diskrete Transitionen werden über das Schlüsselwort **discrete** bekanntgegeben. Auch hier kann in kompakter syntaktischer Form eine Zusammenfassung von Teilbedingungen und Teilmengen von Zuweisungen mit:

[**if** Alternative von (Wächter [**->** Aktion)]
for
{
Liste Diskreter Übergänge
}

erfolgen. Die Wächter enthalten dabei zu empfangende Signale und Bedingungen bzw. die Aktionen die zu sendenden Signale und Zuweisungen, die an allen unter **for** aufgeführten Übergängen auftreten. Ein diskreter Übergang wird in der Form:

leave Liste von Lokationen **reach** Liste von Lokationen
[**if** Alternative von (Wächter [**->** Aktion)]

oder

remain Liste von Lokationen
[**if** Alternative von (Wächter [**->** Aktion)]

spezifiziert. Im ersten Fall werden Lokationen verlassen, um nachfolgende Lokationen zu erreichen. Im zweiten Fall handelt es sich um die Ausführung einer Schleife. Die Beschreibung der Struktur nach dem Schlüsselwort **if** enthält alle zu empfangenden und zu sendenden Signale sowie damit verbundene Bedingungen und Zuweisungen, die ausschließlich für die jeweiligen unter **leave** und **remain** notierten Übergänge gelten. Anhand des Beispiels eines Senders aus der Abbildung 6.12 sollen die syntaktischen Strukturen für kontinuierliche und diskrete Transitionen veranschaulicht werden.

Beispiel 6.3.2
Die kontinuierlichen Abläufe in den Lokationen 'warten', 'senden' und 'wiederholen' werden in MODEL-HS folgendermaßen zusammengefasst:

```
continuous
    while 0 ≤ X
    do X :   X+1
    in warten, senden(while X ≤ λ), wiederholen(while X < 2σ);
```

und die diskreten Übergänge werden wie:

```
discrete
    if receive {beschäftigt}
    for
    {
       remain warten
       if true -> B_B :   1;
       remain senden
       if 0 < X;
       remain wiederholen
       if X < 2σ -> B_B :   1;
    }
    if receive {bereit} -> B_B :   0
    for
    {
       remain warten;
       remain wiederholen
       if X < 2σ;
    }
    if receive {sendet} -> X :   0
    for
    {
       leave warten reach senden
       if B_B    0 -> send {beginn};
       leave warten reach wiederholen
       if B_B    1;
    }
    if true -> X :   0
    for
    {
       leave senden reach wiederholen
       if receive {kollidiert}, X < σ;
```

```
    leave wiederholen reach senden
    if X < 2σ, B_B    0 -> send {beginn};
    leave senden reach warten
    if X    λ -> send {ende};
}
```

formuliert. Die Notation bietet verschiedene Kombinationsmöglichkeiten, deren Nutzung dem Anwender überlassen sind.

Verhalten in VYSMO

In VYSMO wird das *Verhalten eines hierarchisch hybriden Automaten* mit Hilfe von drei verschiedenen Lokationen und Übergängen zwischen diesen Lokationen dargestellt. Die Lokationen lassen sich in:

1. Anfangslokationen,
2. einfache Lokationen und
3. komplexe Lokationen

unterteilen.
Die *Anfangslokation* wird wie in Abbildung 6.15 durch einen schwarzen Punkt gekennzeichnet, der einen Wartezustand ohne damit verbundene Bedingungen und Flussläufe widerspiegelt. Diese Ausgangslokation kann sofort verlassen werden, wenn die Bedingung eines Übergangs von der Anfangslokation zu einer nachfolgenden Lokation erfüllt ist.

Abbildung 6.15: Anfangslokation

Einfache Lokationen, Abbildung 6.16, kennzeichnen kontinuierliche Verläufe, die nicht weiter unterteilt werden können. Für solche Lokationen wird der Name angegeben, eine Invariante, die während des gesamten Aufenthaltes in der Lokation gültig ist und Aktivitäten zur Beschreibung des Fortschreitens von Werten kontinuierlicher Variablen und Uhren in Abhängigkeit von der Zeit.
Komplexe Lokationen abbildung 6.17, kennzeichnen kontinuierliche Verläufe, die weiter verfeinert werden können. Die Lokation bildet in diesem Fall eine Instanz von einem Automaten- oder Blocktyp zur Verfeinerung. Die spezifizierte Invariante und zugehörige Aktivitäten gelten für den gesamten Verlauf. Korrespondierend zu den formalen Parametern des Automaten- bzw. Blocktypen sind aktuelle Parameter bezüglich fester Werte, Variablen, Uhren und Signalen anzugeben. Dabei entscheidet die Reihenfolge über die Zuordnung der aktuellen zu den formalen Parametern. Einfache und komplexe Lokatio-

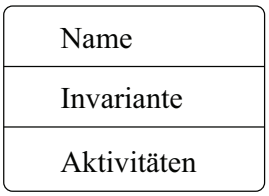


Abbildung 6.16: Einfache Lokation

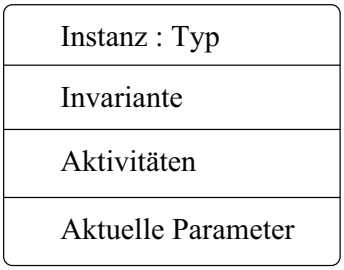


Abbildung 6.17: Komplexe Lokation

nen können als Endlokationen zur Akzeptanz von Wörtern gekennzeichnet werden. Dazu erhalten diese Lokationen eine doppelte Umrandung.
Übergänge zwischen den Lokationen werden wie in Abbildung 6.18 notiert. Der Wächter verbindet zu empfangende Signale und Bedingungen, unter welchen der Übergang ausgeführt werden kann. Die Aktion besteht aus zu sendenden Signalen und Wertzuweisungen zu Uhren und Variablen, die während des Überganges vorgenommen werden.

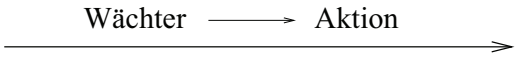


Abbildung 6.18: Übergang

6.3.5 Fortsetzung - Beispielsystem in MODEL-HS und VYSMO

Das im Beispiel 6.2.5 vorgestellte System eines Wasserbehälters basiert auf den 2 hierarchisch hybriden Automaten 'Wasserstand' und 'Schalter'. Für diese Automaten werden hier die textuelle Notation in MODEL-HS und die graphische Notation in VYSMO gegenübergestellt. In MODEL-HS wird der Automat des Wasserstandes wie folgt notiert:

Wasserstand (**invariant** WassInv;
activity WassAct;
parameter Abfluss1, Abfluss2;

```
            signal synchron in   wasser_an,schneller
                          out    pump_an
                  refine  out    pump_aus);
declaration
    variable
        continuous Stand;
    location
        simple zufließen, langsam_pumpen, schnell_pumpen;

behaviour
    initialisation
        if receive{wasser_an} -> Stand :   0;
        location zufließen;
    termination
        zufließen;
    transition
        continuous
            while Stand ≤ 22
            in zufließen(while 0 ≤ Stand; do Stand :   Stand + 1),
               langsam_pumpen(while 12 ≤ Stand; do Stand :   Stand - Abfluss1),
               schnell_pumpen(while 2 ≤ Stand; do Stand :   Stand - Abfluss2);
        discrete
            for
               {
               leave zufließen reach langsam_pummpen
               if Stand > 20 -> send{pump_an};
               leave langsam_pumpen reach schnell_pumpen
               if receive{schneller};
               leave schnell_pumpen reach zufließen
               if Stand < 2 -> send{pump_aus};
               }
endautomaton Wasserstand;
```

Die graphische Umsetzung des Automaten 'Wasserstand' ist in Abbildung 6.19 zu finden. Nachdem der Zufluss in der Umgebung mit 'wasser_an' freigeschaltet wurde, soll solange Wasser in den Behälter gefüllt werden bis ein Wasserstand größer als 20 Einheiten erreicht wurde. Dann kann die Pumpe angeschaltet werden, um wieder Wasser aus dem Behälter herauszupumpen. Entsprechend der Invariante der Lokation 'zufließen' muss die Pumpe jedoch bis zum Wasserstand von 22 Einheiten angeschaltet werden, da die Lokation danach verlassen sein muss. Zu jedem Zeitpunkt bis zum Stand von 12 erreichten Einheiten kann die Pumpe in der Lokation 'langsam_pumpen' durch die Umgebung in einen schnelleren Pumprhythmus versetzt werden. Reagiert die Umgebung nicht mit dem Signal 'schneller' bis zur Wasserstandsuntergrenze von 12 Einheiten, so wird der

Übergang verpasst und der Ablauf kann nicht mehr akzeptiert werden. In der Lokation 'schnell_pumpen' kann bis zu einer Wasserstandsuntergrenze von 2 Einheiten die Pumpe abgeschaltet und die Endlokation 'zufließen' erreicht werden.

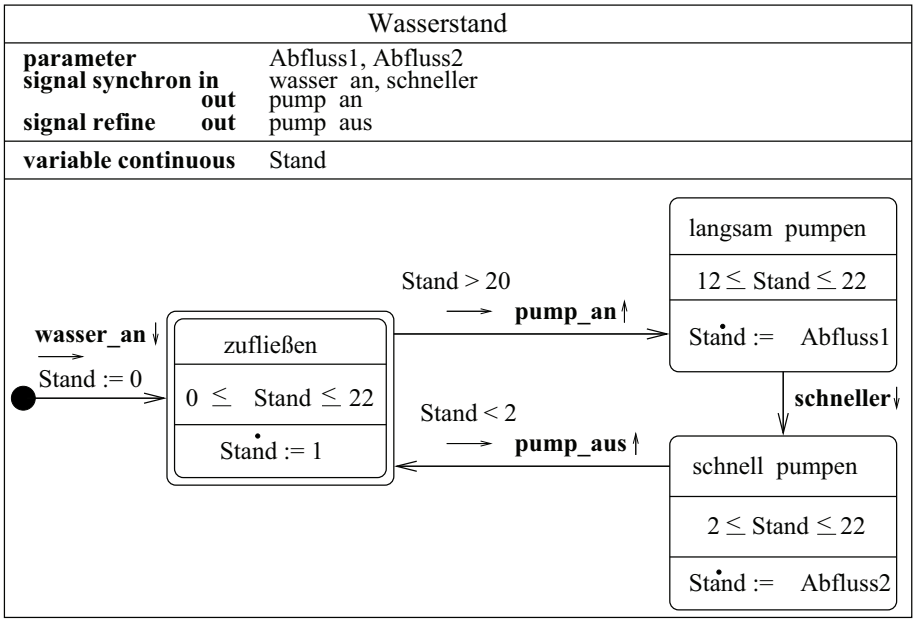


Abbildung 6.19: Wasserstandsanzeige

Der Automat des Schalters lässt sich in MODEL-HS wie folgt notieren:

```
Schalter (invariant SchaltInv;
          activity SchaltAct;
          parameter Leist;
          signal synchron in   an
                        out  voll_L
                 refine      in   aus);
refined from Library by
    Stufen (invariant Inv;
            activity Act;
            parameter L;
            signal synchron out v_Leist
                   refine      in   aus);
declaration
    clock X;
    location
       simple ausgeschaltet;
       complex angeschaltet: Stufen;
```

```
behaviour
    matching
        angeschaltet(0 ≤ X ≤ 30, {X :   X+1}, Leist, voll_L, aus⟩);
    initialisation
        if true -> X :   0;
        location ausgeschaltet;
    termination
        ausgeschaltet;
    transition
        continuous
            while 0 ≤ X
            do X :   X+1
            in ausgeschaltet, angeschaltet(while X ≤ 30);
        discrete
            if true -> X :   0
            for
               {
               leave ausgeschaltet reach angeschaltet
               (if receive{an};
                 leave angeschaltet reach ausgeschaltet
                 if receive{aus};
                 }
endautomaton Schalter;
```

In VYSMO steht die Abbildung 6.20 zur Visualisierung des Schalters zur Verfügung.

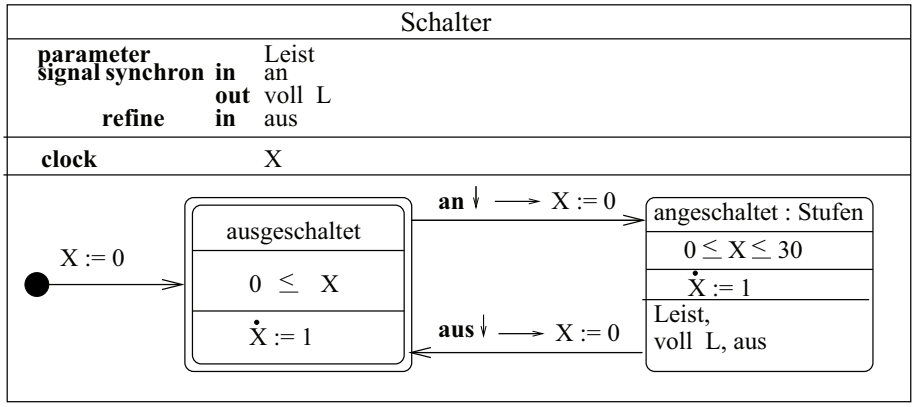


Abbildung 6.20: Pumpe als Schaltelement

Sofort mit dem Start des gesamten Systems 'Wasserbehälter' befindet sich der Automat

'Schalter' in der Lokation 'ausgeschaltet' und die Uhr 'X' wurde mit dem Wert Null initialisiert. Wurde aus der Umgebung, hier in Bezug auf den Wasserstand, das Anschalten der Pumpe mit 'an' angefordert, so wird die Lokation nach 'angeschaltet' gewechselt. Die Grenze der Invariante von 'X $\leq$ 30' besagt, dass diese Lokation spätestens nach 30 Zeiteinheiten unabhängig von einem auslösenden Signal zu verlassen ist. Die Lokation 'angeschaltet' kann als komplexe Lokation verfeinert werden. Hierzu wird ein weiterer hierarchisch hybrider Automat 'Stufen' verwendet, dem als aktuelle Parameter die Leistung der Pumpe 'Leist' und die zwei Signale 'voll_L' und 'aus' übergeben werden. Wird in der Umgebung ein Signal 'aus' zum Ausschalten der Pumpe innerhalb der 30 Zeiteinheiten gesendet, so kann der Automat unter Zurücksetzen der Uhr 'X' auf Null in die Ausgangslokation 'ausgeschaltet', die gleichzeitig eine Endlokation bildet, zurückkehren. Der Automat 'Stufen' bildet eine mögliche Verfeinerung des komplexen Zustandes 'angeschaltet' in dem Automaten 'Schalter'. In MODEL-HS sieht die Notation wie folgt aus:

```
Stufen (invariant StufInv;
          activity StufAct;
          parameter L;
          signal synchron out v_Leist
                 refine     in   aus);
declaration
    clock Y;
    location
       simple Stufe_1, Stufe_2, Stufen_aus;

behaviour
    initialisation
       if true -> Y :   0;
       location Stufe_1;
    termination
       Stufen_aus;
    transition
       continuous
          while 0 ≤ Y
          do Y :   Y+1;
          in Stufe_1(while Y ≤ 10/L), Stufe_2, Stufen_aus;
       discrete
          for
             {
             leave Stufe_1 reach Stufe_2
             if Y > 10/L -> send{v_Leist};
             leave Stufe_2 reach Stufen_aus
             if receive{aus}, Y < 100;
```

}
endautomaton Stufen;

Visualisiert liegt der Automat ’Stufen’ in VYSMO wie in der Abbildung 6.21 vor.

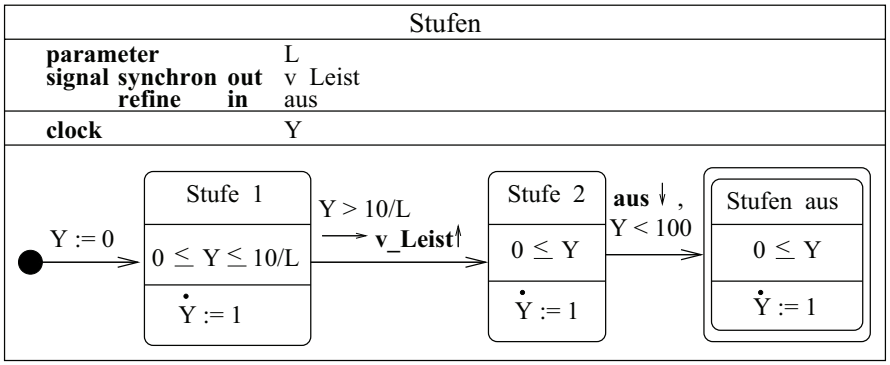


Abbildung 6.21: Stufen des Schaltelementes

Zwei Leistungsstufen werden während des angeschalteten Zustandes der Pumpe genutzt. Die erste Leistungsstufe ’Stufe_1’ liegt sofort nach dem Einschalten der Pumpe an, die zweite Leistungsstufe ’Stufe_2’ soll nach einer Zeit von mindestens ’10/L’ Einheiten eingenommen werden und solange anliegen, bis von der Umgebung ein Signal ’aus’ empfangen wird. Dann ist der gesamte Vorgang des Automaten ’Stufen’ akzeptiert und im Automaten ’Schalter’ wird somit die Endlokation ’ausgeschaltet’ erreicht.

Kapitel 7

Weitere Arbeiten

Im Zusammenhang mit unseren Sprachen MODEL-HS und VYSMO sowie der symbolischen Simulation in CLP sind Arbeiten entstanden, deren Konzepte in der Zukunft für Verbesserungen und Anpassungen zur Verfügung stehen sowie weiterführende Entwicklungen erlauben. Dazu gehören:

das Werkzeug 'ROSSY' (ROStocker SYmbolic Simulation) [TLR04],

die Anfragesprache 'MODEL-HS-Query' mit zugehöriger Übersetzung in unsere temporal-logische Sprache 'SyS-TPTL' (TPTL for Symbolic Simulation) [Sik06, TSR07] bzw. 'VYSMO-Query' [TRB01],

Umsetzung und deren Probleme in nutzerfreundliche Anfragemasken (NUR-Projekt 2005) und

Anwendungen im Bereich von Studiensystemen und Parallelen Programmen [LTR03, TR05, TLR06, BLL+06, BLLT07a, BLLT07b].

In den folgenden Abschnitten sollen diese Ansätze kurz skizziert werden.

7.1 Werkzeug ROSSY

Zur Schaffung von Modellen, Spezifikation von Eigenschaften und Ausführung der symbolischen Simulation in Prolog IV entstand das Konzept des Werkzeuges 'ROSSY'. Im Vorfeld war zur Entwicklung von Modellen in VYSMO ein graphischer Editor [Bra00] in JAVA mit einem Austauschformat in XML zur Sprache MODEL-HS entstanden, welcher in Anbindung an 'ROSSY' genutzt wird.

7.1.1 Architektur

Die Architektur von 'ROSSY' ist in der Abbildung 7.1 dargestellt. Ein hybrides System wird in MODEL-HS als eine Menge nebenläufiger, hybrider Automaten modelliert, welche mit einem auf Korrektheit nachweisbaren Übersetzer in Laptob [LR01] nach Prolog IV, aufgeteilt in eine Symboltabelle mit zugehörigem hybriden System, in Form von CLP Regeln transformiert werden. Die hierarchisch hybriden Automaten werden dabei entsprechend des Pseudocodes aus Anhang A in flache Automaten überführt, da Prolog IV eine hierarchische Modulstruktur fehlt. Die Symboltabelle bildet ein technisches Hilfsmittel zur Vermeidung von Umbenennungen auf Lokationsnamen, Variablen und Signalen. Auf diese Art und Weise bleibt eine Übersichtlichkeit des Prolog IV Programmes für Testzwecke erhalten. Eine Eigenschaft als Anfrage in MODEL-HS wird neben dem

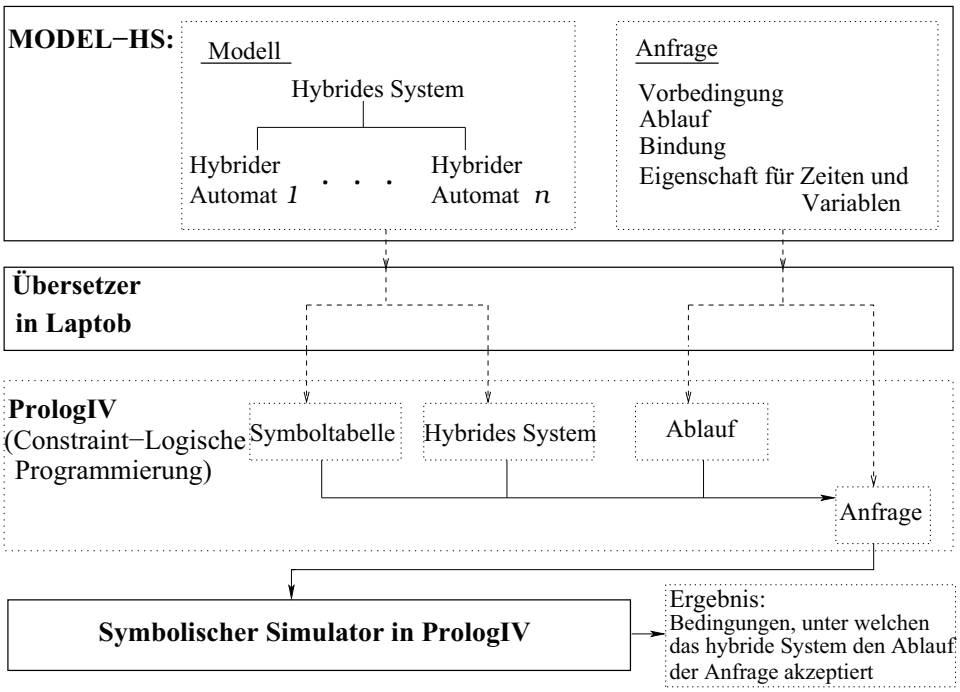


Abbildung 7.1: Architektur von ROSSY

Modell spezifiziert, die:

1. *Vorbedingungen* zum Setzen von Anfangswerten für Parameter und Variablen,

2. den *Ablauf* zum Erreichen einer Eigenschaft in Form eines Zeitwortes,

3. die *Bindung* an Längengrenzen zur Ausführung der symbolischen Simulation wie Anzahl zu durchlaufender Transitionen bzw. Zyklen und

4. vom Nutzer erwünschte *Eigenschaften* für Zeiten der Akzeptanz von Symbolen und Variablen

enthalten kann. Die Anfrage wird in ein Prologziel überführt, wobei der Ablauf wie auch die Symboltabelle und das eigentliche hybride System einen Parameter dieses Anfragezieles bilden. Nach Ausführung der symbolischen Simulation in Prolog IV liegen im Ergebnis Bedingungen vor, die eine Aussage über die Akzeptanz des gegebenen Ablaufes unter den spezifizierten Vorbedingungen, Bindungen und nutzerdefinierten Eigenschaften in dem modellierten, hybriden System zulassen.

7.1.2 Ausführung in Prolog IV

Die Ausführung wurde in Prolog IV vorgenommen, da in dieser Prologversion schon frühzeitig:

1. eine Intervallarithmetik für reellwertige Zahlen, die eine genaue Analyse hybrider Systeme gerade zu Verifikationszwecken zulässt, wie vor allem in [Wit04] mit umfangreichen Untersuchungen belegt wurde,

2. Constraints für unterschiedliche Domänen wie boolsche, numerische Werte und Listen, die zu einer adäquaten Modellierung hybrider Systeme führen und

3. nicht-lineare Constraints zur Beschreibung und Ausführung hybrider Systeme in komplexen Bereichen z.B. naturwissenschaftlicher Phenomäne wie der Biologie [TKRE00]

integriert sind.
Zur Verdeutlichung von Problemen, die sich aus der Arbeit mit Prolog IV ergeben, sind hier kurz einige technische Merkmale aufgeführt. Die exakte Berechnung und Repräsentation von reellwertigen Zahlen wie $\sqrt{2}$ ist praktisch nicht möglich. Zur praktischen Berechnung und Repräsentation ist in Prolog IV ein Löser implementiert, welcher die gegebenen exakten Probleme über Intervalle annähert. Dabei gilt:

- Die Lösung eines Basisconstraints ist *korrekt* und *vollständig*.

- Die Lösung einer Menge von Constraints ist *korrekt*, jedoch *nicht vollständig*.

Nicht vollständig bedeutet: Die Möglichkeit besteht, dass der Prolog IV - Löser nicht entdeckt, dass ein Constraintssystem keine Lösung besitzt. Die *Korrektheit* des Algorithmus garantiert jedoch, für den Fall, ein Constraintsystem besitzt eine Lösung, so wird der Prolog IV - Löser niemals schlussfolgern, dass das System keine Lösung besitzt. Prolog IV gibt Garantien bezüglich des ursprünglichen, exakten Problems gegenüber des für die Berechnung und Darstellung transformierten, angenäherten Problems:

- Wenn ein angenähertes Problem als unlösbar gefunden wurde, so ist auch das zugehörige, exakte Problem unlösbar.

- Wenn ein exaktes Problem eine Lösung hat, so befindet sich diese Lösung unter den Lösungen, die beim Lösen des angenäherten Problems gefunden wurden.

- Wenn für das angenäherte Problem eine Lösung gefunden wurde, für welche alle Variablen einen eindeutigen Wert besitzen, so entspricht diese Lösung der Lösung des exakten Problems.

7.2 Anfragesprache

Standardanfragen der Form:

query (*Eigenschaft, Hybrides System, Startlokation, Startzeit, Startvariablenbelegung*),

wobei die Eigenschaft durch ein gegebenes bzw. aus dem hybriden System berechneten Zeitwort darstellt wird, müssen oft um Umgebungsbedingungen zur symbolischen Simulation erweitert werden. Betrachtet werden dabei:

1. Eingangsbedingungen des gesamten hybriden Systems, die für alle Automaten gelten,

2. Anfangsbedingungen an Startzeit und Startvariablen, um einen Lauf zu konkretisieren,

3. Anzahl zu durchlaufender Transitionen bzw. Zyklen anstelle der Angabe einer Eigenschaft und zur Begrenzung der Ausführungsschritte der symbolischen Simulation sowie

4. harte und weiche Nutzerbedingungen, die bezüglich einzelner Zeiten und Variablenbelegungen eingehalten werden müssen.

In MODEL-HS wurde, wie für das Werkzeug 'ROSSY' bereits aufgeführt, folgende Notation festgelegt:

query <AnfrageName>
  trajectory <Zeitwort>
  commitment <AusführungsGrenzen>
  property <NutzerBedingungen>
endquery.

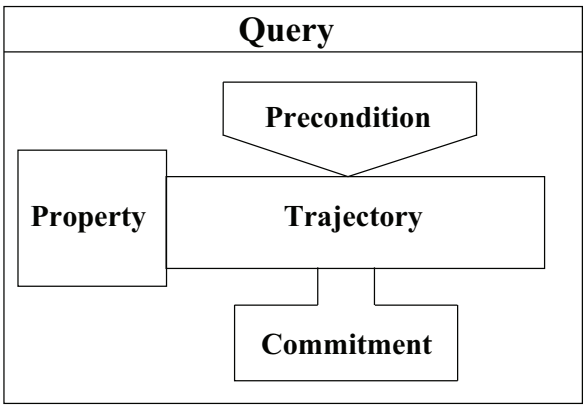


Abbildung 7.2: Anfrage in VYSMO

In [TRB01] wurde die Spezifikation von Anfragen in VYSMO durch graphische Primitiven wie in Abbildung 7.2 visualisiert, deren Bestandteile als Wegweiser für detaillierte Aussagen dienen, die modular über mehrere Ebenen aufgebaut werden können. Trajektorien auf der Basis von Zeitwörtern wurden dabei als Erweiterung von UML-Sequenzdiagrammen geschaffen. Für die Nutzerbedingungen entstanden graphische Darstellungsmöglichkeiten logischer Schlussfolgerungen, die die Kombination und Überlagerung von Domänen für die Zeiten und Variablen widerspiegeln.
Da Experten und Nutzer von Verifikationsanwendungen stärker mit der Spezifikation von Eigenschaften durch Ausdrücke temporaler Logiken als durch Trajektorien mit zusätzlichen Umgebungsbedingungen vertraut sind, wurde in [Sik06, TSR07] ein Übersetzer implementiert, welcher die automatische Transformation von Anfragen, geschrieben in MODEL-HS, zu temporallogischen Ausdrücken einer auf unseren Ansatz angepassten Version der in [AH89, AH93] entwickelten Logik TPTL und zurück ermöglicht.

7.3 Anfragemasken

Nicht in jedem Fall ist ein formaler Ausdruck zur nutzerfreundlichen Spezifikation von Eigenschaften geeignet. Für solche Fälle wurden Überlegungen zur Gestaltung vorgefertigter Anfragemasken in Betracht gezogen. Dabei konnte gezeigt werden, dass formale Spezifikationen zur Vereinfachung von Anfragen und Erreichung einer Nutzerfreundlichkeit um Merkmale der Mehrdeutigkeit zu erweitern sind, die mit zusätzlichen Regeln zur formalen Auswertung verbunden werden müssen. Anhand eines verkürzten Beispiels zum Ablegen von Leistungen eines Studiums der Informatik aus [TLR06] soll die Problematik skizziert werden.

Beispiel 7.3.1
Angenommen, eine Leistung wird mithilfe folgender Grammatik definiert:

<Leistung> :: <Veranstaltung>.<LeistungsArt>.<Versuch>.<Erfolg>
<Veranstaltung>:: PT | SWT | MA
<LeistungsArt> :: PRUEF | LN
<Versuch> :: 1. Versuch | 1. Wdh. | 2. Wdh.
<Erfolg> :: b | nb

Eine Leistung kann dementsprechend in einem Fach Programmierungstechnik 'PT', Softwaretechnik 'SWT' bzw. Mathematik 'MA' mit einer Prüfung 'PRUEF' oder einem Leistungsnachweis 'LN' in einem '1. Versuch', einer ersten Wiederholung '1. Wdh.' bzw. zweiten Wiederholung '2. Wdh.' mit dem Erfolg 'b' (bestanden) bzw. 'nb' (nicht bestanden) abgelegt werden. Formal sind alle Elemente einer Leistung in dieser Art und Weise korrekt spezifiziert worden. Doch für die Nutzung in ausgewählten Anfragen sind Erweiterungen notwendig wie z.B. in folgender Machbarkeitsanfrage.
*Die **Machbarkeit** von Studiensystemen beinhaltet dabei zum einen den Begriff der **Kohärenz**, wie dieser in [STG01] geprägt wurde, um Widersprüche und Lücken des Lehrplanes bezüglich der Studien- und Prüfungsordnung aufzudecken und zum anderen Wünsche für zukünftige Studienabläufe entsprechend der individuellen Ausgestaltung und Lehrplansituation zu überprüfen.*

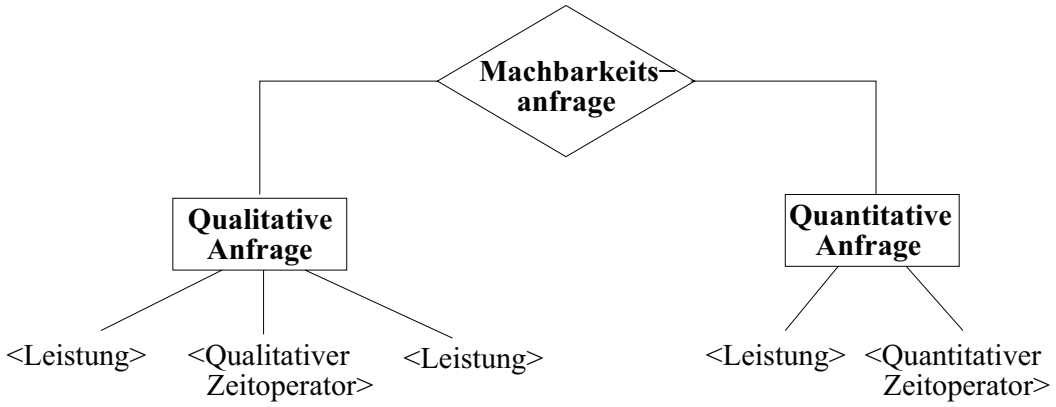


Abbildung 7.3: Machbarkeitsanfragen

Wie in Abbildung 7.3 werden Anfragen nach der Machbarkeit in qualitative und quantitative Anfragen eingeteilt. Bei qualitativen Anfragen wird die Abfolge zweier Leistungen betrachtet, wobei ein qualitativer Zeitoperator mit folgenden Terminalen belegt sein kann:

<QualitativerZeitoperator> :: vor | nach | mit .

Quantitative Anfragen beschäftigen sich mit Zeiträumen, in denen eine Leistung absolviert werden muss bzw. kann. Der qualitative Zeitoperator entspricht dabei folgender Grammatik:

<QuantitativerZeitoperator>	*::*	<Präposition> <Wert> <Einheit>
<Präposition>	*::*	vor | nach | im | innerhalb
<Wert>	*::*	<ReelleZahl>
<Einheit>	*::*	Monat | Semester .

Zur Gestaltung nutzerfreundlicher Masken in Form von Listen wählbarer Terminale ist zu beachten, dass für den Nutzer oft nicht alle theoretisch möglichen Fälle interessant sind. Aus praktischer Sicht sollen Wahlmöglichkeiten zusammenfassbar sein, um die Anzahl notwendiger Nutzereingaben zu reduzieren.

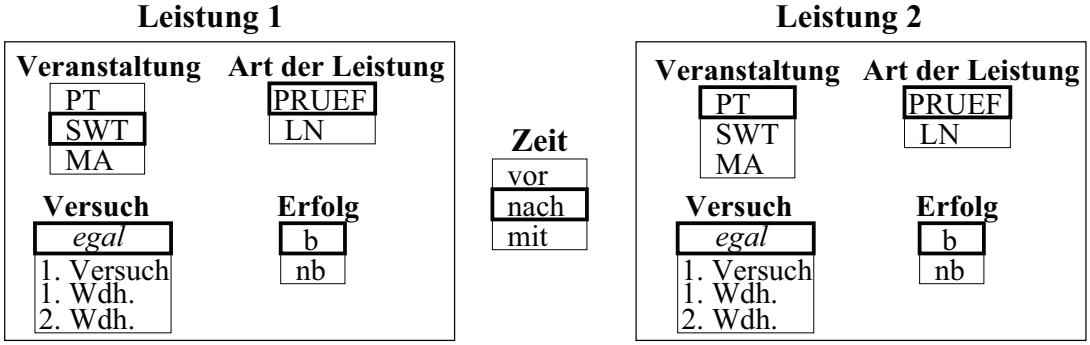


Abbildung 7.4: Maske für Qualitative Anfrage

Die Abbildung 7.4 beinhaltet eine nutzerfreundliche Maske zur Ausführung einer qualitativen Anfrage der Machbarkeit. Die dick umrandeten Felder verweisen dabei auf die ausgewählten Terminale durch den Nutzer, welcher wissen möchte, ob eine Prüfung im Fach 'SWT' unabhängig von welchem Versuch 'nach' einer Prüfung im Fach 'PT' auch unabhängig von welchem Versuch bestanden werden kann. Die Unabhängigkeit des Versuches wird mit dem neuen Terminal 'egal' eingeführt, dessen Semantik entsprechend der folgenden, attributierten Grammatik festgelegt werden kann:

<Leistung> *::* <Veranstaltung>.<LeistungsArt>.<Versuch>.<Erfolg>
*{*if *Val(Versuch)* ← *'egal'* ∧ *Val(Erfolg)* ← *'b'*
then *Val(Leistung)* ←
⟨*Val(Veranstaltung), Val(LeistungsArt), bestand('1. Versuch','1. Wdh.','2. Wdh'), Val(Erfolg)*⟩
elseif *Val(Versuch)* ← *'egal'* ∧ *Val(Erfolg)* ← *'nb'*
then *Val(Leistung)* ←
⟨*Val(Veranstaltung), Val(LeistungsArt), nicht_bestand('1. Wdh.','2. Wdh'), Val(Erfolg)*⟩
else *Val(Leistung)* ←

$\langle Val(Veranstaltung), Val(LeistungsArt),$
$Val(Versuch), Val(Erfolg) \rangle \}$

<Veranstaltung>:: PT $\{Val(Veranstaltung) \leftarrow$ 'PT'$\}$ |
SWT $\{Val(Veranstaltung) \leftarrow$ 'SWT'$\}$ |
MA $\{Val(Veranstaltung) \leftarrow$ 'MA'$\}$

<LeistungsArt> :: PRUEF $\{Val(LeistungsArt) \leftarrow$ 'PRUEF'$\}$ |
LN $\{Val(LeistungsArt) \leftarrow$ 'LN'$\}$

<Versuch> :: egal $\{Val(Versuch) \leftarrow$ 'egal'$\}$ |
1. Versuch $\{Val(Versuch) \leftarrow$ '1. Versuch'$\}$ |
1. Wdh. $\{Val(Versuch) \leftarrow$ '1. Wdh.'$\}$ |
2. Wdh. $\{Val(Versuch) \leftarrow$ '2.Wdh.'$\}$

<Erfolg> :: b $\{Val(Erfolg) \leftarrow$ 'b'$\}$ |
nb $\{Val(Erfolg) \leftarrow$ 'nb'$\}$

Neben bereits bekannten syntaktischen Regeln einer 'Leistung' wird in geschweiften Klammern die Attributierung zu jeder Syntaxregel angegeben. In Hochkommata stehen semantische Werte, die durch den Pfeil auf das Attribut des Nichtterminals der linken Seite der Syntaxregel abgebildet werden. Mithilfe des synthetisierten Attributes 'Val' kann der Wert einer Leistung berechnet werden, auch wenn der Nutzer den Versuch nicht explizit festgelegt hat, sondern durch 'egal' eine mehrdeutige Entscheidung trifft. Ist das Terminal 'egal' im Zusammenhang mit einem erfolgreichen Abschluss 'b' gewählt worden, so kann über die Funktion 'bestand' während der Ausführung der Anfrage nacheinander ermittelt werden, welcher der Versuche '1. Versuch', '1 Wdh.' bzw. '2. Wdh.' als erster zum Erfolg 'b' führt. Im Fall eines erfolglosen Abschlusses 'nb' wird über die Funktion 'nicht_bestand' überprüft, ob der Versuch in der '1. Wdh.' oder in der '2. Wdh' nicht bestanden werden konnte. In welcher Wiederholung das Nichtbestehen einer Prüfung bzw. eines Leistungsnachweises auftreten kann, hängt z.B. von einer möglichen Zulassung zu einer zweiten Wiederholung ab, die als Information in einer gesonderten Datenbank bereit liegen kann.

Aus dem Beispiel ist erkennbar, dass die Entwicklung nutzerfreundlicher Anfragemasken zur Erweiterung zugrundeliegender formaler Grammatiken um Terminale mit mehrdeutigem Informationsgehalt führen. Diese Mehrdeutigkeit kann erst zur Ausführungszeit der Anfrage durch die in den Attributen spezifizierten Funktionen zur Beschreibung der dynamischen Semantik beseitigt werden.

7.4 Anwendung

Hybride Systeme auf Grundlage hybrider Automaten wurden erfolgreich auf den Gebieten von Studiensystemen und parallelen Programmen angewendet. Hier konnte gezeigt werden, dass die Verbindung aus einem System von Bedingungen, die unbedingt eingehalten werden müssen und einem System aus Bedingungen, die eingehalten werden

können, zu problemadäquaten Modellen in Bereichen der Praxis führen, in denen zwei verschiedene Arten von Anforderungen unter Echtzeitbedingungen von Bedeutung sind. Durch die Verwendung hybrider Systeme können die Anforderungen voneinander vollständig getrennt analysiert und modelliert und danach in einem Modell hybrider Automaten elegant ohne Zwischentransformationen vereinigt und auf Konsistenz sowie weitere Eigenschaften überprüft werden. Weiterhin kann die symbolische Simulation durch die Integration domain-spezifischer Aspekte zur Lösung von Constraint-Systemen effizienter ausgeführt werden, da in Abhängigkeit vom Anwendungsgebiet Bedingungen gegenüber anderen Bedingungen mit höherer Priorität auftreten können. Die höhere Priorität kann aus praktischer Sicht in ausgewählten Fällen zur vollständigen Überdeckung von Bedingungen mit niedriger Priorität führen und somit zur Vermeidung von:

- Konflikten auf mathematischer Ebene,
- zusätzlichen Vereinfachungen und
- Konsistenzprüfungen.

Die Erarbeitung solcher Bedingungen in verschiedenen Domänen und deren Auswirkungen auf die symbolische Simulation werden Gegenstand zukünftiger Arbeiten sein.

7.4.1 Studiensysteme

In [TR05, TLR06] wurden Studiensysteme als Anwendungsbereich für hybride Systeme näher betrachtet. Auf der einen Seite liegen solchen Systemen zeitliche Beschränkungen zum Ablegen von Nachweisen aus Studien- und Prüfungsordnungen vor und auf der anderen Seite legen Durchführungsbestimmungen, individuelle Auslegungen durch Dozenten sowie Angebote an Lehrveranstaltungen die konkrete Ausgestaltung der zeitlichen Beschränkungen fest. Zeitliche Beschränkungen der Studien- und Prüfungsordnungen bilden unbedingte Grenzen, die zum Ablegen von Leistungen notwendigerweise eingehalten werden *müssen*. Hinreichend *können* Leistungen jedoch nur unter den zeitlichen Beschränkungen aus Durchführungsbestimmungen, individuellen Auslegungen durch Dozenten sowie Angeboten an Lehrveranstaltungen abgelegt werden. Die beiden Systeme aus notwendigen und hinreichenden Bedingungen können getrennt voneinander formalisiert werden und wie in Abbildung 7.5 als eine Einheit in hybriden Systemen auf:

a) Konsistenz
b) Machbarkeit

überprüft werden.
Konsistenz ist dabei eine Eigenschaft, die sich nur auf die Studien- und Prüfungsordnung einer Studienrichtung bezieht. Mit dieser Eigenschaft soll gewährleistet sein, dass

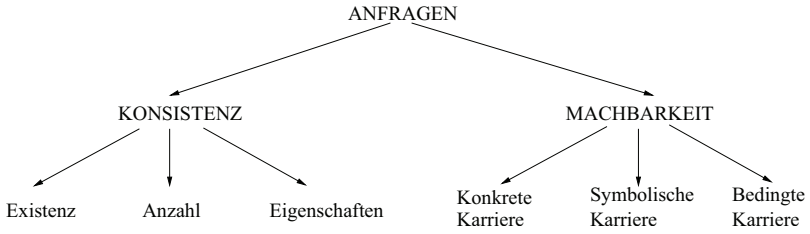


Abbildung 7.5: Struktur spezifizierbarer Anfragen

die Studien- und Prüfungsordnung keine Widersprüche aufweist und das Studium entsprechend dieser Ordnung, welche auch von Umgebungsbedingungen wie Rahmenordnungen abhängt, erfolgreich abgeschlossen werden kann. Konsistenz bildet den wesentlichen Indikator zum Test, ob Neufassungen, Änderungen oder Erweiterungen der Studien- und Prüfungsordnung die Eigenschaften der Widerspruchsfreiheit und erfolgreicher Absolvierung des Studiums erhalten.

Begriff 7.4.1
Eine Studien- und Prüfungsordnung heißt **konsistent**, wenn genau diejenigen Studienabläufe akzeptiert werden, die gegebenen Konsistenzbedingungen entsprechen.

Hier stellen sich folgende Fragen:

Existenz:	Gibt es Studienkarrieren, die innerhalb übergeordneter Beschränkungen die Bedingungen der Studien- und Prüfungsordnung erfüllen?
Anzahl:	Wieviele Studienkarrieren erfüllen die Bedingungen der Studien- und Prüfungsordnung innerhalb übergeordneter Beschränkungen?
Eigenschaften:	Welcher Art sind diese Studienkarrieren?

Die **Machbarkeit** von Studiensystemen besteht wie bereits im vorhergehenden Abschnitt angesprochen in der Kohärenz zur Aufdeckung von Widersprüchen und Lücken in Lehrplänen bezüglich von Studien- und Prüfungsordnungen und zur Überprüfung von Wünschen für zukünftige Studienabläufe entsprechend der individuellen Ausgestaltung und Lehrplansituation. Aufgrund dieser begrifflichen Festlegung setzt sich die Machbarkeit mit Fragestellungen zu ausgewählten Studienverläufen auseinander. Im diesem Zusammenhang sind 3 Fragen bezüglich eines gegebenen Leistungsstandes, einer Studien- und Prüfungsordnung, der Ausgestaltung von Regeln durch Dozenten und angebotener Lehrveranstaltungen zu beantworten:

Konkrete Karriere:	Ist eine Folge von Leistungen, wobei jede Leistung mit dem exakten Zeitpunkt des Ablegens dieser Leistung verbunden ist, machbar?
Symbolische Karriere:	Ist eine Folge von Leistungen, wobei jede Leistung mit einem

	symbolischen Zeitpunkt des Ablegens dieser Leistung verbunden ist, machbar?
Bedingte Karriere:	Ist eine Folge von Leistungen in Form von Bedingungen spezifizierter Wünsche machbar?

Die Überprüfung der Anfragen kann in Abhängigkeit der beiden Systeme von Bedingungen, wobei die Bedingungen aus Studien- und Prüfungsordnungen gegenüber Bedingungen aus Durchführungsbestimmungen, individuellen Auslegungen durch Dozenten sowie Angeboten an Lehrveranstaltungen mit höherer Priorität einzuordnen sind, ausgeführt werden.

7.4.2 Parallele Programme - Softwareverifikation

Wie in [BLL+06, BLLT07a, BLLT07b] untersucht, kann auch zur Verifikation paralleler Programme unter Echtzeitbetrachtung eine Einteilung notwendiger und hinreichender Bedingungen vorgenommen werden. Diese Unterscheidung wird in Fällen wirksam, in welchen unbedingt zu gewährleistende Nutzeranforderungen an die Software, z.B. Kundenwünsche, die in Lasten- und Pflichtenheften festgehalten werden, hinreichenden Bedingungen gegebener Hardwarekonfigurationen zur Ausführung der Programme gegenüberstehen. Software- und Hardwareanforderungen bilden zwei voneinander unabhängige Systeme von Bedingungen, die oft in verschiedenen Gebieten der Informatik, wie Softwaretechnik und Technische Informatik, analysiert, im Entwurf integriert und zur Ausführung betrachtet werden. Sollen jedoch Anfragen wie:

1. Machbarkeit von Softwareanforderungen unter gegebenen Hardwarekonfigurationen,
2. Änderung von Hardwarekonfigurationen zur Erfüllung gegebener Softwareanforderungen und
3. Suche nach nutzerfreundlichen, effizienten Kompromisslösungen zur Anpassung von Software- und Hardwareanforderungen

gestellt werden, die beide Systeme von Bedingungen betreffen, so kann die Überprüfung bequem in einem hybriden System durchgeführt werden. Welche Bedingungen der Software- bzw. Hardwareanforderungen mit einer höheren Priorität betrachtet werden, hängt vom Verwendungszweck der Ergebnisse ab. Hierbei ist z.B. entscheidend, ob:

- und in welchem Maße Softwareanforderungen anpassbar sind bzw.
- Hardwarekonfigurationen aus wirtschaftlichen Gründen bestehen bleiben müssen.

Kapitel 8

Zusammenfassung und Ausblick

Die Arbeit ist durch die Entwicklung der deklarativen, modularen Sprachen MODEL-HS und VYSMO zur nutzerfreundlichen Beschreibung hybrider Systeme, die der symbolischen Simulation in CLP dienen, im Umfeld eines breiten Spektrums an Einflussfaktoren gekennzeichnet. In diesem Rahmen werden die erreichten Ergebnisse und zukünftige Untersuchungen in den folgenden Abschnitten dargestellt.

8.1 Erreichtes

Die Arbeit kann unter folgenden Punkten zusammengefasst werden:

Grundlagen:

- Beschreibung hybrider Systeme über flachen Strukturen,
- Definition von Zeitwörtern bezüglich hybrider Automaten,
- Untersuchung der symbolischen Simulation als Prozess der Akzeptanz von Zeitwörtern in hybriden Automaten,
- Ausführung der symbolischen Simulation in CLP,

Entwicklung:

- Formalisierung deklarativer und modularer Strukturen zur Bildung von Hierarchien bezüglich Verfeinerung und Synchronisation bezüglich Abstraktion über hybriden Automaten zur symbolischen Simulation,
- Beschreibung der Sprachen MODEL-HS und VYSMO, dessen Notation der Darstellung des implementierten graphischen Editors entspricht,
- Konzeption des Werkzeuges ROSSY,
- Transformation von Zeitwörtern als temporal-logische Ausdrücke,
- Ideen nutzerfreundlicher Anfragemasken,

Anwendung:

- Modellierung von Studiensystemen über Studien- und Prüfungsordnungen sowie Durchführungsbestimmungen, individuellen Auslegungen durch Dozenten und Angeboten an Lehrveranstaltungen,
- Transformation paralleler Programme in hybride Systeme zur Verifikation zeitkritischer Eigenschaften in Bezug auf Softwareanforderungen und Hardwarebedingungen, die die Ausführungszeit betreffen.

Verbunden mit der Erarbeitung dieser Punkte ist die Einordnung der entwickelten Sprachen in den Bereich bestehender Sprachen zur Beschreibung hybrider Systeme, die Abgrenzung der symbolischen Simulation zur klassischen Simulation und zur Verifikation sowie der Vergleich der symbolischen Simulation zum Bounded Model Checking.

8.2 Verbesserung der Ausführungszeit

Die Nutzung der Sprachkonzepte von MODEL-HS und VYSMO im Werkzeug ROSSY als auch die Ansätze, welche im Umfeld der Schaffung von nutzerfreundlichen Beschreibungen hybrider Systeme zur symbolischen Simulation in CLP entstanden sind, lassen einen breiten Raum für zukünftige Untersuchungen und Entwicklungen zu.
Für die symbolische Simulation ist die Leistungsfähigkeit, welche durch den Einsatz effizienter Methoden zur Ausführung und die Nutzung modularer Strukturen in der Modellierung erreicht werden kann, noch nicht vollständig ausgeschöpft. Aus technischer Sicht sind zur Verbesserung der Ausführungszeit der symbolischen Simulation:

1. leistungsfähige Constraintlöser mit beschleunigenden Methoden zur Ausführung wie Backjumping und Konfliktlernen [FH05] einzusetzen,
2. Ansätze priorisierter Constraints mit Blick auf das Anwendungsgebiet [BLL+06] zu nutzen,
3. Überführung hierarchisch hybrider Automaten in flache Automaten zu vermeiden, indem Läufe als geschachtelte Zeitwörter basierend auf [ACM06, AM06, AAB+07] abgearbeitet werden,
4. Schlussfolgerungstechniken über modularen Strukturen [HK97] wie annahme - garantiertes Schlussfolgern (assume-guarantee reasoning) [HMP01, Rus01, AG04] in unsere Methode zu integrieren bzw.
5. Lösbarkeit der Constraint-Systeme aus den Kombinationen von Invarianten, Aktivitäten, Wächtern und Aktionszuweisungen, die aus Verfeinerung und Synchronisation hervorgehen, als Voraussetzung entsprechend wohldefinierter Modi aus [AGLS06] zu fordern und bereits während der Übersetzung nach CLP zu überprüfen.

8.3 Vervollständigung und Erweiterung

Im Zusammenhang mit der Entwicklung von MODEL-HS und VYSMO lassen die breit gefächerten Untersuchungen Fragen zur Vollständigkeit und Erweiterbarkeit von Ergebnissen offen. Dazu gehören folgende Problemstellungen:

- Für die transitive Hülle zur Berechnung von Abhängigkeiten zwischen Automaten in Bezug auf die Synchronisation ist bisher der einfachste Fall betrachtet worden, bei welchem sämtliche an einer Synchronisation beteiligten Automaten höchstens mit einer Transition beteiligt sein können. Diese Beziehungen sind vereinfacht. Um praktisch relevante Fälle wie im Ansatz der Sprache 'Shift' [DGS98, DGV97] zu erfassen, in denen alternativ verschiedene Transitionen hybrider Automaten zu einer Synchronisation beitragen können, muss der Algorithmus der transitiven Hülle erweitert werden.

- Da aus entwurfstechnischer Sicht für unsere Sprachen MODEL-HS und VYSMO zur Schaffung der Übersichtlichkeit und guten Wiederverwendung die beliebige Schachtelung von HHAs und SHAs als SHHAs zugelassen und somit zur Modellierung die in [AKY99] definierte 'Wohlstrukturiertheit' in unseren hierarchisch hybriden Automaten aufgehoben, jedoch zur symbolischen Simulation aus Gründen der Vereinfachung die Wohlstrukturiertheit erhalten wurde, ist in der Zukunft eine Kompromisslösung zur Beseitigung dieser Unterschiede zu schaffen.

- Bisher betreffen die neuen Verfeinerungs- und Abstraktionstechniken bezüglich der Akzeptanz von Zeitwörtern hauptsächlich die Verhaltens- und Synchronisationsbeschreibungen der hybriden Automaten. Zukünftig ist zu betrachten, inwiefern Anforderungen wohldefinierter Schnittstellen bezüglich von Verfeinerung und Abstraktion hybrider Automaten eingehalten werden müssen. Zum Beispiel sollen feste, nicht änderbare Parameter nur an feste nicht änderbare Parameter übergeben werden bzw. das Einfrieren von Uhren und Werten kontinuierlicher Variablen kann durch die Übergabe an feste, nicht änderbare Parameter unter vorgegebenen Bedingungen erfolgen.

- In MODEL-HS und VYSMO ist im Gegensatz zu SDL die dynamische Typisierung von Automaten zulässig. Probleme, die im Zusammenhang mit einer dynamisch zu ändernden Struktur im Verhalten bzw. der Synchronisation auftreten können, sind näher zu untersuchen. Gerade zur Erstellung komplexer Systeme sind Techniken zur Verfügung zu stellen, durch die Seiteneffekte zu vermeiden sind.

- Zwischen den Ergebnissen der Entscheidbarkeit und Erreichbarkeit, welche auf dem Gebiet der hybriden Systeme durch verschiedene Ansätze zur Analyse errungen werden, kann mit Entscheidbarkeitsaussagen im Bereich der Sprachen über attributierten Grammatiken, die durch die symbolische Simulation hybrider Automaten erreicht werden, eine enge Verbindung geschaffen werden. Dadurch lassen sich Ergebnisse der beiden Bereiche vergleichen und aufeinander übertragen.

- Nutzerfreundliche Beschreibungen von Anfragen und nutzerfreundliche Anfragemasken müssen in der Zukunft durch aussagekräftige Beispiele stärker validiert werden.

- Die Ergebnisse der symbolischen Simulation sind entsprechend des Anwendungsgebietes auszuwerten und nutzerfreundlich aufzubereiten. Hier ist die Schaffung einer Klassifikation entsprechend anwendungstypischer Eigenschaften denkbar.

- Das Werkzeug ROSSY ist in seiner Gesamtheit zu implementieren und mit Bezug auf die genannten Ergebnisse zu erweitern.

Anhang A

Rekursive Variante der Transformation eines HHA in einen flachen Automaten

define function flat_HHA ($HHA, SysInv, SysAct, ActualPara$)
let $HHA \quad \langle Name, Interface, \langle Simple, Complex, \langle l_{wait}, \delta_0, l_0,$
$TF, FF, Clock, Var, Guard, Action, Inv, Act\rangle\rangle\rangle$ **in**
clean(transform($HHA, SysInv, SysAct, ActualPara$),$FF$);

define function transform($\langle Name, Interface, \langle Simple,$
$\langle KL, \delta_{ek}, \delta_{ke}, \delta_{kk}, HHAs, Actual\rangle, General\rangle,$
$SysInv, SysAct, ActualPara$)
if $KL \quad \{\}$ **then**
simple_case($Name, Interface, Simple, General, SysInv, SysAct,$
$ActualPara, \langle\emptyset, \emptyset, \emptyset,$noname,$\emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset,$
Id,Tautology$\rangle$);
else
let $l \quad$ first(KL) **and** $R \quad$ rest(KL) **and**
$\langle \delta_{lek}, \delta_{lke}, \delta_{lkk}\rangle \quad$ transitions($\delta_{ek}, \delta_{ke}, \delta_{kk}, l$) **and**
$General \quad \langle l_{wait}, \delta_0, l_0, TF, FF, Clock, Var, Guard, Action, Inv, Act\rangle$ **and**
$HHAs(l) \quad \langle lName, lInterface, \langle lSimple, lComplex, \langle ll_{wait}, l\delta_0, ll_0, TF,$
$lFF, lClock, lVar, lGuard, lAction, lInv, lAct\rangle\rangle\rangle$ **in**
concat(clean(transform($HHAs(l), Inv(l), Act(l), Actual(l)), lFF$),
transform($\langle Name, Interface, \langle Simple,$
$\langle R, \delta_{ek}, \delta_{ke}, \delta_{kk}, HHAs, Actual\rangle, General\rangle\rangle,$
$SysInv, SysAct, ActualPara$),
$l, \delta_{lek}, \delta_{lke}, \delta_{lkk}, Interface, General, SysInv, SysAct, ActualPara$);

define function simple_case($Name, \langle HHAInv, HHAAct, Para, SClock, SVar,$
$SynchronSig, RefineSig\rangle,$
$\langle EL, \delta_{ee}\rangle,$

$\langle l_{wait}, \delta_0, l_0, TF, FF, Clock, Var, Guard, Action, Inv, Act\rangle$,
$SysInv, SysAct, ActualPara$,
$\langle fL, fS, fR, fl_0, fTF, fFF, fPara, fClock, fVar,$
$fGuard, fAction, f\delta, fInv, fAct\rangle$)

let $\langle fL, fl_0, fTF, fFF, loPara, loClock, loVar, fInv, fAct\rangle$
simple_locations($Name, EL, l_{wait}, \delta_0, l_0, TF, FF, HHAInv, HHAAct, Para,$
$SClock, Clock, SVar, Var, Inv, Act, SysInv, SysAct, ActualPara,$
$fL, fl_0, fTF, fFF, fPara, fClock, fVar, fInv, fAct$) **in**
let $\langle fS, fR, fPara, fClock, fVar, fGuard, fAction, f\delta\rangle$
simple_transitions($Name, \delta_{ee}, SynchronSig, RefineSig, Para, SClock, Clock,$
$Svar, Var, Guard, Action, ActualPara,$
$fS, fR, loPara, loClock, loVar, fGuard, fAction, f\delta$) **in**
$\langle fL, fS, fR, fl_0, fTF, fFF, fPara, fClock, fVar, fGuard, fAction,$
$f\delta, fInv, fAct\rangle$;

define function simple_locations($Name, EL, l_{wait}, \delta_0, l_0, TF, FF, HHAInv, HHAAct, Para,$
$SClock, Clock, SVar, Var, Inv, Act, SysInv, SysAct, ActualPara,$
$fL, fl_0, fTF, fFF, fPara, fClock, fVar, fInv, fAct$) :
$\langle fL, fl_0, fTF, fFF, fPara, fClock, fVar, fInv, fAct\rangle$;
begin
if $HHAInv$ blank **then**
$SysInv$: true
endif;
if $HHAAct$ {} **then**
$SysAct$: {}
endif;
for all $l \in EL$ **do**
if $\delta_0 \neq \langle l_{wait}, \langle\{\}, C\rangle, \langle\{\}, A\rangle, l_0\rangle \wedge l = l_{wait}$ **then**
$fl_0 := l_{wait}$;
fL : $fL \cup \{l\}$;
if $\delta_0 = \langle l_{wait}, \langle\{\}, C\rangle, \langle S, A\rangle, l_0\rangle$ **then**
$fInv(l)$: (0 $\leq$ value($SystemClock$) SystemClockIn) $\wedge$ $SysInv$;
else
$fInv(l)$: (0 $\leq$ value($SystemClock$)) $\wedge$ $SysInv$;
endif;
$fAct(l)$: {value($SystemClock$) : value($SystemClock$)+ takt($SystemClock$)} $\cup$ $SysAct$;
$fClock$: $fClock \cup \{SystemClock\}$
else
if δ_0 $\langle l_{wait}, \langle\{\}, C\rangle, \langle\{\}, A\rangle, l_0\rangle \wedge l = l_0$ **then**
$fl_0 := l_0$
endif;
if $l \neq l_{wait}$ **then**
if $l \in TF$ **then**

$fTF := fTF \cup \{l\}$
endif;
if $l \in FF$ **then**
$fFF := fFF \cup \{l\}$
endif;
$fL := fL \cup \{l\}$;
$fInv(l)$: replace_para_in_cond($Inv(l), Para, SClock, SVar, ActualPara$) $\wedge$ $SysInv$;
$fAct(l)$: replace_para_in_assign($Act(l), Para, SClock, SVar, ActualPara$) $\cup$ $SysAct$;
$fPara$: $fPara \cup$
replace_para(para_in_cond($Inv(l), Para$) $\cup$ para_in_assign($Act(l), Para$),
$ActualPara$);
$fClock$: $fClock \cup$
replace_clock(clock_in_Cond($Inv(l), SClock, Clock$) $\cup$
clock_in_assign($Act(l), SClock, Clock$),
$Sclock, ActualPara$);
$fVar$: $fVar \cup$
replace_var(var_in_cond($Inv(l), SVar, Var$) $\cup$
var_in_assign($Act(l), SVar, Var$),
$SVar, ActualPara$);
endif;
endif;
end for all;
end function;

function simple_transitions($\delta_{ee}, SynchronSig, RefineSig, Para, SClock, Clock, Svar, Var,$
$Guard, Action,$
$ActualPara,$
$fS, fR, loPara, loClock, loVar, fGuard, fAction, f\delta$) :
$\langle fS, fR, fPara, fClock, fVar, fGuard, fAction, f\delta \rangle$;
begin
for all $\langle k, \langle GRec, GCond \rangle, \langle ASend, AFunct \rangle, l \rangle \in \delta_{ee}$ **do**
if $GRec \neq \emptyset \vee ASend \neq \emptyset$ **then**
$f\delta$: $f\delta \cup \langle k, \langle$replace_sig($GRec, SynchronSig, RefineSig, ActualPara$),
replace_para_in_cond($GCond, Para, SClock, SVar, ActualPara$)$\rangle$,
$\langle$replace_sig($ASend, SynchronSig, RefineSig, ActualPara$),
replace_para_in_funct($AFunct, Para, SClock, SVar, ActualPara$)$\rangle$,
$l\rangle$;
$fPara$: $fPara \cup$ replace_para(para_in_cond($GCond, Para$) $\cup$
para_in_funct($AFunct, Para$),$ActualPara$);
$fClock$: $fClock \cup$ replace_clock(clock_in_cond($GCond, SClock, Clock$) $\cup$
clock_in_funct($AFunct, SClock, Clock$),
$SClock, ActualPara$);

$fVar$: $fVar \cup$ replace_var(var_in_cond($GCond, SVar, Var$) $\cup$
var_in_funct($AFunct, SVar, Var$),SVar,ActualPara);
fR : $fR \cup$ replace_sig($GRec, SynchronSig, RefineSig, ActualPara$);
fS : $fS \cup$ replace_sig($ASend, SynchronSig, RefineSig, ActualPara$);
endif;
end for all
end function;

function concat($\langle lL, lS, lR, ll_0, lTF, lFF, lPara, lClock, lVar, lGuard, lAction, l\delta, lAct, lInv\rangle$,
$\langle fL, fS, fR, fl_0, fTF, fFF, fPara, fClock, fVar, fGuard, fAction, f\delta,$
$fInv, fAct\rangle$,
$l, \delta_{ek}, \delta_{ke}, \delta_{kk}, \langle HHAInv, HHAAct, Para, SClock, SVar,$
$SynchronSig, RefineSig\rangle$,
$\langle l_{wait}, \delta_0, l_0, TF, FF, Clock, Var, Guard, Action, Inv, Act\rangle$,
$SysInv, SysAct, ActualPara$) :
$\langle fL, fS, fR, fl_0, fTF, fFF, fPara, fClock, fVar, fGuard, fAction, f\delta,$
$fInv, fAct\rangle$;
begin
$\langle fL, fl_0, fTF, fFF, fPara, fClock, fVar, fInv, fAct\rangle$:
concat_locations($lL, ll_0, lTF, lFF, lPara, lClock, lVar, lAct, lInv,$
$fL, fl_0, fTF, fFF, fPara, fClock, fVar, fInv, fAct,$
$l, HHAInv, HHAAct, Para, SClock, SVar, l_0, FF, Clock, Var,$
$SysInv, SysAct, ActualPara$);
$\langle fS, fR, fPara, fClock, fVar, fGuard, fAction, f\delta\rangle$:
concat_transitions($, ll_0, lTF, lFF, lPara, lClock, lVar, l\delta,$
$fS, fR, fPara, fClock, fVar, fGuard, fAction, f\delta,$
$l, \delta_{ek}, \delta_{ke}, \delta_{kk}, SynchronSig, RefineSig,$
$\delta_0, Para, SClock, SVar, Clock, Var, Guard, Action, ActualPara$);
end function;

function concat_locations($lL, ll_0, lTF, lFF, lPara, lClock, lVar, lAct, lInv,$
$fL, fl_0, fTF, fFF, fPara, fClock, fVar, fInv, fAct,$
$l, HHAInv, HHAAct, Para, SClock, SVar, l_0, FF, Clock, Var,$
$SysInv, SysAct, ActualPara$) :
$\langle fL, fl_0, fTF, fFF, fPara, fClock, fVar, fInv, fAct\rangle$;
begin
if $HHAInv$ blank **then**
$SysInv$: true
endif;
if $HHAAct$ {} **then**
$SysAct$: {}
endif;
if l l_0 **then**

fl_0 : $l.ll_0$
endif;
if $l \in FF$ **then**
fFF : $fFF \cup \{l.ll_0\}$
endif;
for all $m \in lL$ **do**
if $m \in lTF$ **then**
fTF : $fTF \cup \{l.m\}$
else
fL : $fL \cup \{l.m\}$
if $m \in lFF$ **then**
fFF : $fFF \cup \{l.m\}$
endif;
$fInv(l.m)$: replace_para_in_cond(
rename_ident_in_cond($lInv(m), lPara, lClock, lVar, l$),
$Para, SClock, SVar, ActualPara$) $\wedge$ $SysInv$;
$fAct(l.m)$: replace_para_in_assign(
rename_ident_in_assign($lAct(m), lPara, lClock, lVar, l$),
$Para, SClock, SVar, ActualPara$) $\cup$ $SysAct$;
$fPara$: $fPara \cup$
replace_para(
search_para(rename_ident(para_in_cond($lInv(m), lPara$) $\cup$
para_in_assign($lAct(m), lPara$),l),
$Para$),
$ActualPara$);
$fClock$: $fClock \cup$
replace_clock(
search_clock(rename_ident(para_in_cond($lInv(m), lPara$) $\cup$
para_in_assign($lAct(m), lPara$),l),
$SClock, Clock$) $\cup$
rename_ident(lclock_in_cond($lInv(m), lClock$) $\cup$
lclock_in_assign($lAct(m), lClock$),l),
$SClock, ActualPara$);
$fVar$: $fVar \cup$
replace_var(
search_var(rename_ident(para_in_cond($lInv(m), lPara$) $\cup$
para_in_assign($lAct(m), lPara$),l),
$SVar, Var$) $\cup$
rename_ident(lvar_in_cond($lInv(m), lVar$) $\cup$
lvar_in_assign($lAct(m), lVar$),l),
$SVar, ActualPara$);
endif;
end for all

end function;

function concat_transitions($ll_0, lTF, lFF, lPara, lClock, lVar, l\delta$,
$fS, fR, fPara, fClock, fVar, fGuard, fAction, f\delta$,
$l, \delta_{ek}, \delta_{ke}, \delta_{kk}, SynchronSig, RefineSig$,
$\delta_0, Para, SClock, SVar, Clock, Var, Guard, Action, ActualPara$) :
$\langle fS, fR, fPara, fClock, fVar, fGuard, fAction, f\delta\rangle$;
begin
for all $\langle m, \langle GRec, GCond\rangle, \langle ASend, AFunct\rangle, n\rangle \in l\delta$ **do**
if ($\langle GRec, ASend\rangle \neq \langle\{\}, \{\}\rangle$) **then**
$f\delta$: $f\delta \cup$
$\langle l.m,$
$\langle$replace_sig($GRec, SynchronSig, RefineSig, ActualPara$),
replace_para_in_cond(
rename_ident_in_cond($GCond, lPara, lClock, lVar, l$),
$Para, SClock, SVar, ActualPara$)
$\rangle$,
$\langle$replace_sig($ASend, SynchronSig, RefineSig, ActualPara$),
replace_para_in_assign(
rename_ident_in_funct($AFunct, lPara, lClock, lVar, l$),
$Para, SClock, SVar, ActualPara$)
$\rangle$,
$l.n\rangle$
fR : $fR \cup$ replace_sig($GRec, SynchronSig, RefineSig, ActualPara$);
fS : $fS \cup$ replace_sig($ASend, SynchronSig, RefineSig, ActualPara$);
$fPara$: $fPara \cup$
replace_para(
search_para(rename_ident(para_in_cond($GCond, lPara$) $\cup$
para_in_funct($AFunct, lPara$),l),
$Para$),
$ActualPara$);

$fClock$: $fClock \cup$
replace_clock(
search_clock(rename_ident(para_in_cond($GCond, lPara$) $\cup$
para_in_funct($AFunct, lPara$),l),
$SClock, Clock$) $\cup$
rename_ident(lclock_in_cond($GCond, lClock$) $\cup$
lclock_in_funct($AFunct, lClock$),l),
$SClock, ActualPara$);
$fVar$: $fVar \cup$
replace_var(
search_var(rename_ident(para_in_cond($GCond, lPara$) $\cup$

para_in_funct($AFunct, lPara$),l),
$SVar, Var$) ∪
rename_ident(lvar_in_cond($GCond, lVar$) ∪
lvar_in_funct($AFunct, lVar$),l),
$SVar, ActualPara$);

$fGuard$: $fGuard$ ∪
{⟨replace_sig($GRec, SynchronSig, RefineSig, ActualPara$),
replace_para_in_cond(
rename_ident_in_cond($GCond, lPara, lClock, lVar, l$),
$Para, SClock, SVar, ActualPara$)⟩};

$fAction$: $fAction$ ∪
{⟨replace_sig($ASend, SynchronSig, RefineSig, ActualPara$),
replace_para_in_assign(
rename_ident_in_funct($AFunct, lPara, lClock, lVar, l$),
$Para, SClock, SVar, ActualPara$⟩};

endif;
end for all;
for all ($d = \langle j, \langle GRec, GCond\rangle, \langle ASend, AFunct\rangle, k\rangle$) $\in \delta_{ek} \cup \delta_{ke} \cup \delta_{kk}$ **do**
if $d \in \delta_{ek}$ **then**
$f\delta$: $f\delta$ ∪
⟨j,
⟨replace_sig($GRec, SynchronSig, RefineSig, ActualPara$),
replace_para_in_cond($GCond, Para, SClock, SVar, ActualPara$)
⟩,
⟨replace_sig($ASend, SynchronSig, RefineSig, ActualPara$),
replace_para_in_funct($AFunct, Para, SClock, SVar, ActualPara$)
⟩,
$l.ll_0$⟩
$fPara$: $fPara$ ∪ replace_para(para_in_cond($GCond, Para$) ∪
para_in_funct($AFunct, Para$),$ActualPara$);
$fClock$: $fClock$ ∪ replace_clock(clock_in_cond($GCond, SClock, Clock$) ∪
clock_in_funct($AFunct, SClock, Clock$),
$SClock, ActualPara$);

$fVar$: $fVar$ ∪ replace_var(var_in_cond($GCond, SVar, Var$) ∪
var_in_funct($AFunct, SVar, Var$),$SVar, ActualPara$);
$fGuard$: $fGuard$ ∪
{⟨replace_sig($GRec, SynchronSig, RefineSig, ActualPara$),
replace_para_in_cond($GCond, Para, SClock, SVar, ActualPara$)⟩};
$fAction$: $fAction$ ∪
{⟨replace_sig($ASend, SynchronSig, RefineSig, ActualPara$),
replace_para_in_funct($AFunct, Para, SClock, SVar, ActualPara$)⟩}
endif;

if $(d \in \delta_{ke}) \vee (d \in \delta_{kk} \wedge j = l)$ **then**
for all $((t = \langle m, \langle tGRec, tGCond\rangle, \langle tASend, tAFunct\rangle, n\rangle) \in l\delta) \wedge (n \in lFF \cup lTF)$ **do**
if vmatch$(GRec, tGRec, GCond, tGCond, ASend, tASend, AFunct, tAFunct)$ **then**
if $d \in \delta_{ke}$ **then**
$f\delta :\ f\delta \cup$
$\langle j.m,$
$\langle$replace_sig$(GRec, SynchronSig, RefineSig, ActualPara)$,
replace_para_in_cond(
rename_ident_in_cond$(tGCond, lPara, lClock, lVar, l) \wedge GCond$,
$Para, SClock, SVar, ActualPara)$
$\rangle$,
$\langle$replace_sig$(ASend, SynchronSig, RefineSig, ActualPara)$,
replace_para_in_funct(
rename_ident_in_funct$(tAFunct, lPara, lClock, lVar, l) \cup \{AFunct\}$,
$Para, SClock, SVar, ActualPara)$
$\rangle$,
$k\rangle$
endif;
if $d \in \delta_{kk}$ **then**
for all $k.loc \in fL$ **do**
if notexists$(\langle k.front, G, A, k.loc\rangle, f\delta)$ **then**
$k.l_0 :\ k.loc$;
endif;
end for all;
$f\delta :\ f\delta \cup$
$\langle j.m,$
$\langle$replace_sig$(GRec, SynchronSig, RefineSig, ActualPara)$,
replace_para_in_cond(
rename_ident_in_cond$(tGCond, lPara, lClock, lVar, l) \wedge GCond$,
$Para, SClock, SVar, ActualPara)$
$\rangle$,
$\langle$replace_sig$(ASend, SynchronSig, RefineSig, ActualPara)$,
replace_para_in_funct(
rename_ident_in_funct$(tAFunct, lPara, lClock, lVar, l) \cup \{AFunct\}$,
$Para, SClock, SVar, ActualPara)$
$\rangle$,
$k.l_0\rangle$
endif;
$fPara :\ fPara \cup$
replace_para(
search_para(
rename_ident(para_in_cond$(tGCond, lPara) \cup$
para_in_funkt$(tAFunct, lPara)$,l),

$Para$) ∪ para_in_cond($GCond, Para$) ∪ para_in_funct($AFunct, Para$), $ActualPara$);

$fClock$: $fClock$ ∪
replace_clock(
search_clock(
rename_ident(para_in_cond($tGCond, lPara$) ∪
para_in_funkt($tAFunct, lPara$),l),
$SClock, Clock$) ∪
rename_ident(lclock_in_cond($tGCond, lClock$) ∪
lclock_in_funkt($tAFunct, lClock$),l) ∪
clock_in_cond($GCond, SClock, Clock$) ∪
clock_in_funkt($AFunct, SClock, Clock$),$SClock, ActualPara$);

$fVar$: $fVar$ ∪
replace_var(
search_var(
rename_ident(para_in_cond($tGCond, lPara$) ∪
para_in_funct($tAFunct, lPara$),l),
$SVar, Var$) ∪
rename_ident(lvar_in_cond($tGCond, lVar$) ∪
lvar_in_funct($tAFunct, lVar$),l) ∪
var_in_cond($GCond, SVar, Var$) ∪ var_in_funct($AFunct, SVar, Var$),
$SVar, ActualPara$);

$fGuard$: $fGuard$ ∪
{⟨replace_sig($GRec, SynchronSig, RefineSig, ActualPara$),
replace_para_in_cond(
rename_ident_in_cond($tGCond, lPara, lClock, lVar, l$) ∧ $GCond$,
$Para, SClock, SVar, ActualPara$)⟩};

$fAction$: $fAction$ ∪
{⟨replace_sig($ASend, SynchronSig, RefineSig, ActualPara$),
replace_para_in_funct(
rename_ident_in_funct($tAFunct, lPara, lClock, lVar, l$) ∪ {$AFunct$},
$Para, SClock, SVar, ActualPara$)⟩}

endif;
end for all;
endif;
if ($d \in \delta_{kk}$) ∧ ($k = l$) **then**
for all $t = \langle k.Before, \langle tGRec, tGCond\rangle, \langle tASend, tAFunct\rangle, k.Loc\rangle) \in f\delta$) ∧
($k.Loc \in fFF \cup fTF$) **do**
if nmatch($GRec, tGRec, GCond, tGCond, ASend, tASend, AFunct, tAFunct$) **then**
$f\delta$: $f\delta$ ∪
⟨$k.Before$,
⟨$GRec$,

$tGCond \wedge$ replace_para_in_cond($GCond, Para, SClock, SVar, ActualPara$)
$\rangle$,
$\langle ASend$,
$tAFunct \cup$ {replace_para_in_funct($AFunct, Para, SClock, SVar,$
$ActualPara$)}
$\rangle$,
$l.ll_0\rangle$
$fPara$: $fPara \cup$
replace_para(
para_in_cond($GCond, Para$) $\cup$ para_in_funct($AFunct, Para$),
$ActualPara$);

$fClock$: $fClock \cup$
replace_clock(
clock_in_cond($GCond, SClock, Clock$) $\cup$
clock_in_funct($AFunct, SClock, Clock$),$SClock, ActualPara$);
$fVar$: $fVar \cup$
replace_var(
var_in_cond($GCond, SVar, Var$) $\cup$ var_in_funct($AFunct, SVar, Var$),
$SVar, ActualPara$);
$fGuard$: $fGuard \cup$
$\{\langle GRec$,
$tGCond \wedge$
replace_para_in_cond($GCond, Para, SClock, SVar, ActualPara$)$\rangle\}$;
$fAction$: $fAction \cup$
$\{\langle ASend$,
$tAFunct \cup$
{replace_para_in_funct($AFunct, Para, SClock, SVar, ActualPara$)}$\rangle\}$
endif;
end for all;
endif;
fR : $fR \cup$ replace_sig($GRec, SynchronSig, RefineSig, ActualPara$);
fS : $fS \cup$ replace_sig($ASend, SynchronSig, RefineSig, ActualPara$);
end for all;
end function;

function clean ($\langle fL, fS, fR, fl_0, fTF, fFF, fPara, fClock, fVar,$
$fGuard, fAction, f\delta, fInv, fAct\rangle, FF$) :
$\langle fL, fS, fR, fl_0, fTF, fFF, fPara, fClock, fVar,$
$fGuard, fAction, f\delta, fInv, fAct\rangle$;
begin
for all $l.Rest \in fTF$ **do**
fTF : $fTF \setminus l.Rest$;

for all $t = \langle Loc, G, A, l.Rest \rangle \in f\delta$ **do**
$f\delta : \quad f\delta \setminus t$
end for all;
end for all;
for all $l.Rest \in fFF \wedge l \notin FF$ **do**
$fFF : \quad fFF \setminus l.Rest$
end for all;
end function;

Anhang B

Iterativ-rekursive Variante der Transformation eines HHA in einen flachen Automaten

function flat_hha ($HHA, SysInv, SysAct, ActualPara$) : $Automaton$;

begin
$Automaton$: trans_HHA($HHA, SysInv, SysAct, ActualPara$);
end function;

function trans_HHA($\langle Name, Interface, \langle Simple, Complex, General\rangle\rangle$,
$SysInv, SysAct, ActualPara$) : $Automaton$;

begin
$SimpleAutomaton$: simple_case($Interface, Simple, General, SysInv, SysAct,$
$ActualPara, \langle\emptyset, \emptyset, \emptyset,$,noname,,$\emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset,$Id,Tautology$\rangle$);
$Automaton$: complex_case($Interface, Complex, General, SysInv, SysAct, ActualPara,$
$SimpleAutomaton$);
end function;

function simple_case($\langle HHAInv, HHAAct, Para, SClock, SVar, SynchronSig, RefineSig\rangle$,
$\langle EL, \delta_{ee}\rangle$,
$\langle l_{wait}, \delta_0, l_0, TF, FF, Clock, Var, Guard, Action, Inv, Act\rangle$,
$SysInv, SysAct, ActualPara$,
$\langle fL, fS, fR, fl_0, fTF, fFF, fPara, fClock, fVar,$
$fGuard, fAction, f\delta, fInv, fAct\rangle$) :
$\langle fL, fS, fR, fl_0, fTF, fFF, fPara, fClock,$
$fVar, fGuard, fAction, f\delta, fInv, fAct\rangle$;
begin

$\langle fL, fl_0, fTF, fFF, loPara, loClock, loVar, fInv, fAct\rangle$:
simple_locations($EL, l_{wait}, l_0, \delta_0, TF, FF, HHAInv, HHAAct, Para, SClock, Clock,$
$SVar, Var, Inv, Act, SysInv, SysAct, ActualPara,$
$fL, fl_0, fTF, fFF, fPara, fClock, fVar, fInv, fAct$);
$\langle fS, fR, fPara, fClock, fVar, fGuard, fAction, f\delta\rangle$:
simple_transitions($\delta_{ee}, \delta_0, SynchronSig, RefineSig, Para, SClock, Clock, Svar, Var,$
$Guard, Action,$
$ActualPara,$
$fS, fR, loPara, loClock, loVar, fGuard, fAction, f\delta$);
end function;

function complex_case($\langle HHAInv, HHAAct, Para, SClock, SVar, SynchronSig, RefineSig\rangle$,
$\langle KL, \delta_{ek}, \delta_{ke}, \delta_{kk}, HHAs, Actual\rangle$,
$\langle l_{wait}, \delta_0, l_0, TF, FF, Clock, Var, Guard, Action, Inv, Act\rangle$,
$SysInv, SysAct, ActualPara,$
$\langle fL, fS, fR, fl_0, fTF, fFF, fPara, fClock, fVar, fGuard,$
$fAction, f\delta, fInv, fAct\rangle$) :
$\langle fL, fS, fR, fl_0, fTF, fFF, fPara, fClock, fVar, fGuard,$
$fAction, f\delta, fInv, fAct\rangle$;
begin
$\langle\langle fL, fl_0, fTF, fFF, loPara, loClock, loVar, fInv, fAct\rangle, FlatHHA\rangle$:
complex_locations($KL, HHAs, Actual, l_0, FF, HHAInv, HHAAct, Para, SClock, Clock,$
$SVar, Var, Inv, Act, SysInv, SysAct, ActualPara,$
$fL, fl_0, fTF, fFF, fPara, fClock, fVar, fInv, fAct$);
$\langle fS, fR, fPara, fClock, fVar, fGuard, fAction, f\delta\rangle$:
complex_transitions($\delta_{ek}, \delta_{ke}, \delta_{kk}, SynchronSig, RefineSig,$
$\delta_0, Para, SClock, Clock, SVar, Var, Guard, Action,$
$ActualPara,$
$fS, fR, loPara, loClock, loVar, fGuard, fAction, f\delta, FlatHHA$);
end function;

function simple_locations($EL, l_{wait}, l_0, \delta_0, TF, FF, HHAInv, HHAAct, Para, SClock, Clock,$
$SVar, Var, Inv, Act, SysInv, SysAct, ActualPara,$
$fL, fl_0, fTF, fFF, fPara, fClock, fVar, fInv, fAct$) :
$\langle fL, fl_0, fTF, fFF, fPara, fClock, fVar, fInv, fAct\rangle$;
begin
if $HHAInv$ blank **then**
$SysInv$: true
endif;
if $HHAAct$ {} **then**
$SysAct$: {}
endif;
for all $l \in EL$ **do**

if $\delta_0 \neq \langle l_{wait}, \langle\{\}, C\rangle, \langle\{\}, A\rangle, l_0\rangle \wedge l = l_{wait}$ **then**
$fl_0 := l_{wait}$;
fL: $fL \cup \{l\}$;
if δ_0 $\langle l_{wait}, \langle\{\}, C\rangle, \langle S, A\rangle, l_0\rangle$ **then**
$fInv(l)$: (0 ≤ value($SystemClock$) SystemClockIn) ∧ $SysInv$;
else
$fInv(l)$: (0 ≤ value($SystemClock$)) ∧ $SysInv$;
endif;
$fAct(l)$: {value($SystemClock$): value($SystemClock$)+ takt($SystemClock$)} ∪ $SysAct$;
$fClock$: $fClock \cup \{SystemClock\}$
else
if δ_0 $\langle l_{wait}, \langle\{\}, C\rangle, \langle\{\}, A\rangle, l_0\rangle \wedge l = l_0$ **then**
$fl_0 := l_0$
endif;
if $l \neq l_{wait}$ **then**
if $l \in TF$ **then**
$fTF := fTF \cup \{l\}$
endif;
if $l \in FF$ **then**
$fFF := fFF \cup \{l\}$
endif;
$fL := fL \cup \{l\}$;
$fInv(l)$: replace_para_in_cond($Inv(l), Para, SClock, SVar, ActualPara$) ∧ $SysInv$;
$fAct(l)$: replace_para_in_assign($Act(l), Para, SClock, SVar, ActualPara$) ∪ $SysAct$;
$fPara$: $fPara \cup$
replace_para(para_in_cond($Inv(l), Para$) ∪ para_in_assign($Act(l), Para$),
ActualPara);
$fClock$: $fClock \cup$
replace_clock(clock_in_cond($Inv(l), SClock, Clock$) ∪
clock_in_assign($Act(l), SClock, Clock$),
Sclock,ActualPara);
$fVar$: $fVar \cup$
replace_var(var_in_cond($Inv(l), SVar, Var$) ∪
var_in_assign($Act(l), SVar, Var$),
SVar,ActualPara);
endif;
endif;
end for all;
end function;

function simple_transitions($\delta_{ee}, SynchronSig, RefineSig, Para, SClock, Clock, Svar, Var$,
$Guard, Action$,
$ActualPara$,

$fS, fR, loPara, loClock, loVar, fGuard, fAction, f\delta)$:
$\langle fS, fR, fPara, fClock, fVar, fGuard, fAction, f\delta\rangle$;
begin
for all $\langle k, \langle GRec, , GCond\rangle, \langle ASend, , AFunct\rangle, l\rangle \in \delta_{ee}$ **do**
if $GRec \neq \emptyset \vee ASend \neq \emptyset$ **then**
$f\delta$: $f\delta \cup \langle k, \langle$replace_sig($GRec, SynchronSig, RefineSig, ActualPara$),
replace_para_in_cond($GCond, Para, SClock, SVar, ActualPara$)$\rangle$,
$\langle$replace_sig($ASend, SynchronSig, RefineSig, ActualPara$),
replace_para_in_funct($AFunct, Para, SClock, SVar, ActualPara$)$\rangle$,
$l\rangle$;
$fPara$: $fPara \cup$ replace_para(para_in_cond($GCond, Para$) $\cup$
para_in_funct($AFunct, Para$),ActualPara);
$fClock$: $fClock \cup$ replace_clock(clock_in_cond($GCond, SClock, Clock$) $\cup$
clock_in_funct($AFunct, SClock, Clock$),
$SClock, ActualPara$);

$fVar$: $fVar \cup$ replace_var(var_in_cond($GCond, SVar, Var$) $\cup$
var_in_funct($AFunct, SVar, Var$),$SVar, ActualPara$);
fR : $fR \cup$ replace_sig($GRec, SynchronSig, RefineSig, ActualPara$);
fS : $fS \cup$ replace_sig($ASend, SynchronSig, RefineSig, ActualPara$);
endif
end for all
end function;

function complex_locations($KL, HHAs, Actual, l_0, FF, HHAInv, HHAAct, Para, SClock,$
$Clock, SVar, Var, Inv, Act, SysInv, SysAct, ActualPara,$
$fL, fl_0, fTF, fFF, fPara, fClock, fVar, fInv, fAct$) :
$\langle\langle fL, fl_0, fTF, fFF, fPara, fClock, fVar, fInv, fAct\rangle,$
$FlatHHA\rangle$;
begin
if $HHAInv$ blank **then**
$SysInv$: true
endif;
if $HHAAct$ $\{\}$ **then**
$SysAct$: $\{\}$
endif;
for all $l \in KL$ **do**
$FlatHHA[l]$: flat_hha($HHAs(l), Inv(l), Act(l), Actual(l)$;
if l l_0 **then**
fl_0 : $FlatHHA[l].l_0$
endif;
if $l \in FF$ **then**
fFF : $fFF \cup \{FlatHHA[l].\, l_0\}$

endif;
for all $m \in FlatHHA[l].L \setminus FlatHHA[l].TF$ **do**
fL : $fL \cup \{l.m\}$
if $l \in FF \wedge m \in FlatHHA[l].FF$ **then**
fFF : $fFF \cup \{l.m\}$
endif;
$fInv(l.m)$: replace_para_in_cond(
rename_ident_in_cond($FlatHHA[l].Inv(m), FlatHHA[l].Para,$
$FlatHHA[l].Clock, FlatHHA[l].Var, l$),
$Para, SClock, SVar, ActualPara) \wedge SysInv$;
$fAct(l.m)$: replace_para_in_assign(
rename_ident_in_assign($FlatHHA[l].Act(m), FlatHHA[l].Para,$
$FlatHHA[l].Clock, FlatHHA[l].Var, l$),
$Para, SClock, SVar, ActualPara) \cup SysAct$;
$fPara$: $fPara \cup$
replace_para(
search_para(
rename_ident(para_in_cond($FlatHHA[l].Inv(m), FlatHHA[l].Para$) $\cup$
para_in_assign($FlatHHA[l].Act(m), FlatHHA[l].Para$),*l*),
$Para$),
$ActualPara$);
$fClock$: $fClock \cup$
replace_clock(
search_clock(
rename_ident(para_in_cond($FlatHHA[l].Inv(m), FlatHHA[l].Para$) $\cup$
para_in_assign($FlatHHA[l].Act(m), FlatHHA[l].Para$),*l*),
$SClock, Clock$) $\cup$
rename_ident(lclock_in_cond($FlatHHA[l].Inv(m), FlatHHA[l].Clock$) $\cup$
lclock_in_assign($FlatHHA[l].Act(m), FlatHHA[l].Clock$),*l*),
$SClock, ActualPara$);
$fVar$: $fVar \cup$
replace_var(
search_var(
rename_ident(para_in_cond($FlatHHA[l].Inv(m), FlatHHA[l].Para$) $\cup$
para_in_assign($FlatHHA[l].Act(m), FlatHHA[l].Para$),*l*),
$SVar, Var$) $\cup$
rename_ident(lvar_in_cond($FlatHHA[l].Inv(m), FlatHHA[l].Var$) $\cup$
lvar_in_assign($FlatHHA[l].Act(m), FlatHHA[l].Var$),*l*),
$SVar, ActualPara$);
end for all
end for all;
end function;

function complex_transitions($\delta_{ek}, \delta_{ke}, \delta_{kk}, SynchronSig, RefineSig,$
$\delta_0, Para, SClock, Clock, SVar, Var, Guard, Action,$
$ActualPara,$
$fS, fR, fPara, fClock, fVar, fGuard, fAction, f\delta, FlatHHA$) :
$\langle fS, fR, fPara, fClock, fVar, fGuard, fAction, f\delta\rangle$;
begin
if ($\delta_0 = \langle l_{wait}, G, A, l_0\rangle$) $\in \delta_{ek}$ **then**
for all $l \in$ index($FlatHHA$) **do**
with $FlatHHA[l]$ **do**
for all $\langle m, \langle GRec, GCond\rangle, \langle ASend, AFunct\rangle, n\rangle \in FlatHHA[l].\delta$ **do**
if ($\langle GRec, ASend\rangle \neq \langle\{\}, \{\}\rangle$) $\wedge$ ($n \notin FlatHHA[l].TF$) **then**
$f\delta$: $f\delta \cup$
$\langle l.m,$
$\langle$replace_sig($GRec, SynchronSig, RefineSig, ActualPara$),
replace_para_in_cond(
rename_ident_in_cond($GCond, FlatHHA[l].Para,$
$FlatHHA[l].Clock, FlatHHA[l].Var, l$),
$Para, SClock, SVar, ActualPara$)
$\rangle$,
$\langle$replace_sig($ASend, SynchronSig, RefineSig, ActualPara$),
replace_para_in_funct(
rename_ident_in_funct($AFunct, FlatHHA[l].Para,$
$FlatHHA[l].Clock, FlatHHA[l].Var, l$),
$Para, SClock, SVar, ActualPara$)
$\rangle$,
$l.n\rangle$
fR : $fR \cup$ replace_sig($GRec, SynchronSig, RefineSig, ActualPara$);
fS : $fS \cup$ replace_sig($ASend, SynchronSig, RefineSig, ActualPara$);
$fPara$: $fPara \cup$
replace_para(
search_para(
rename_ident(para_in_cond($GCond, FlatHHA[l].Para$) $\cup$
para_in_funct($AFunct, FlatHHA[l].Para$),l),
$Para$),
$ActualPara$);

$fClock$: $fClock \cup$
replace_clock(
search_clock(
rename_ident(para_in_cond($GCond, FlatHHA[l].Para$) $\cup$
para_in_funct($AFunct, FlatHHA[l].Para$),l),
$SClock, Clock$) $\cup$
rename_ident(lclock_in_cond($GCond, FlatHHA[l].Clock$) $\cup$

lclock_in_funct($AFunct, FlatHHA[l].Clock$),l),
$SClock, ActualPara$);
$fVar$: $fVar \cup$
replace_var(
search_var(
rename_ident(para_in_cond($GCond, FlatHHA[l].Para$) $\cup$
para_in_funct($AFunct, FlatHHA[l].Para$),l),
$SVar, Var$) $\cup$
rename_ident(lvar_in_cond($GCond, FlatHHA[l].Var$) $\cup$
lvar_in_funct($AFunct, FlatHHA[l].Var$),l),
$SVar, ActualPara$);
endif;
end for all;
endwith;
end for all;
for all ($d = \langle k, \langle GRec, GCond\rangle, \langle ASend, AFunct\rangle, l\rangle) \in \delta_{ek} \cup \delta_{ke} \cup \delta_{kk}$ **do**
if $d \in \delta_{ek}$ **then**
$f\delta$: $f\delta \cup$
$\langle k,$
$\langle$replace_sig($GRec, SynchronSig, RefineSig, ActualPara$),
replace_para_in_cond($GCond, Para, SClock, SVar, ActualPara$)
$\rangle$,
$\langle$replace_sig($ASend, SynchronSig, RefineSig, ActualPara$),
replace_para_in_funct($AFunct, Para, SClock, SVar, ActualPara$)
$\rangle$,
$FlatHHA(l).FlatHHA[l].l_0\rangle$
$fPara$: $fPara \cup$ replace_para(para_in_cond($GCond, Para$) $\cup$
para_in_funct($AFunct, Para$),ActualPara);
$fClock$: $fClock \cup$ replace_clock(clock_in_cond($GCond, SClock, Clock$) $\cup$
clock_in_funct($AFunct, SClock, Clock$),
$SClock, ActualPara$);

$fVar$: $fVar \cup$ replace_var(var_in_cond($GCond, SVar, Var$) $\cup$
var_in_funct($AFunct, SVar, Var$),$SVar, ActualPara$)
else
for all (($t = \langle m, \langle tGRec, tGCond\rangle, \langle tASend, tAFunct\rangle, n\rangle) \in FlatHHA(k).FlatHHA[l].\delta$)
($n \in FlatHHA(k).FlatHHA[l].FF \cup FlatHHA(k).FlatHHA[l].TF$) **do**
if match($GRec, tGRec, GCond, tGCond, ASend, tASend, AFunct, tAFunct$) **then**
if $d \in \delta_{ke}$ **then**
$f\delta$: $f\delta \cup$
$\langle k.m,$
$\langle$replace_sig($GRec, SynchronSig, RefineSig, ActualPara$),
replace_para_in_cond(

(rename_ident_in_cond($tGCond, FlatHHA[l].Para,$
$FlatHHA[l].Clock, FlatHHA[l].Var, l) \wedge GCond,$
$Para, SClock, SVar, ActualPara$)
$\rangle$,
$\langle$replace_sig($ASend, SynchronSig, RefineSig, ActualPara$),
replace_para_in_funct(
(rename_ident_in_funct($tAFunct, FlatHHA[l].Para,$
$FlatHHA[l].Clock, FlatHHA[l].Var, l) \cup$
$\{AFunct\}$,
$Para, SClock, SVar, ActualPara$)
$\rangle$,
$l\rangle$
endif;
if $d \in \delta_{kk}$ **then**
$f\delta$: $f\delta \cup$
$\langle k.m,$
$\langle$replace_sig($GRec, SynchronSig, RefineSig, ActualPara$),
replace_para_in_cond(
(rename_ident_in_cond($tGCond, FlatHHA[l].Para,$
$FlatHHA[l].Clock, FlatHHA[l].Var, l) \wedge GCond,$
$Para, SClock, SVar, ActualPara$)
$\rangle$,
$\langle$replace_sig($ASend, SynchronSig, RefineSig, ActualPara$),
replace_para_in_funct(
(rename_ident_in_funct($tAFunct, FlatHHA[l].Para,$
$FlatHHA[l].Clock, FlatHHA[l].Var, l) \cup$
$\{AFunct\}$,
$Para, SClock, SVar, ActualPara$)
$\rangle$,
$FlatHHA(l).FlatHHA[l].l_0\rangle$
$fPara$: $fPara \cup$
replace_para(
search_para(
rename_ident(para_in_cond($tGCond, FlatHHA[l].Para$) $\cup$
para_in_funct($tAFunct, FlatHHA[l].Para$),l),
$Para$) $\cup$ para_in_cond($GCond, Para$) $\cup$ para_in_funct($AFunct, Para$),
$ActualPara$);

$fClock$: $fClock \cup$
replace_clock(
search_clock(
rename_ident(para_in_cond($tGCond, FlatHHA[l].Para$) $\cup$
para_in_funct($tAFunct, FlatHHA[l].Para$),l),

$SClock, Clock$) ∪
rename_ident(lclock_in_cond($tGCond, FlatHHA[l].Clock$) ∪
lclock_in_funct($tAFunct, FlatHHA[l].Clock$),$l$) ∪
clock_in_cond($GCond, SClock, Clock$) ∪
clock_in_funct($AFunct, SClock, Clock$),
$SClock, ActualPara$);
$fVar$: $fVar$ ∪
replace_var(
search_var(
rename_ident(para_in_cond($tGCond, FlatHHA[l].Para$) ∪
para_in_funct($tAFunct, FlatHHA[l].Para$),l),
$SVar, Var$) ∪
rename_ident(lvar_in_cond($tGCond, FlatHHA[l].Var$) ∪
lvar_in_funct($tAFunct, FlatHHA[l].Var$),l) ∪
var_in_cond($GCond, SVar, Var$) ∪ var_in_funct($AFunct, SVar, Var$),
$SVar, ActualPara$);
endif;
endif;
fR : fR ∪ replace_sig($GRec, SynchronSig, RefineSig, ActualPara$);
fS : fS ∪ replace_sig($ASend, SynchronSig, RefineSig, ActualPara$);
end for all;
end function;

Anhang C

Berechnung der transitiven Hülle

define function transitive_closure($HHAs_i, HHAsDiff, GR, Cond, AS, Assign,$
$Connect$)
let $HHAs_{i+1} = HHAs_i \cup HHAsDiff$ **in**
if $HHAs_{i+1} = HHAs_i$ **then**
($HHAs_i, GR, Cond, AS, Assign$)
else
let $HHAsDiff = [DiffFirst|DiffRest]$ **in**
transitive_closure($HHAs_{i+1}$,
for_all_diff($DiffRest$,
new_diff(synchron($Connect, DiffFirst,$
$GR[DiffFirst],$
$AS[DiffFirst], HHAs_{i+1}$),
$DiffFirst, \emptyset, GR, Cond, AS, Assign$),
$Connect, HHAs_{i+1}$),
$Connect$);

define function for_all_diff($HHAsDiff, NewDiff, GR, Cond, AS, Assign,$
$Connect, HHAs_{i+1}$)
if $HHAsDiff = \emptyset$ **then**
($NewDiff, GR, Cond, AS, Assign$)
else
let $HHAsDiff = [DiffFirst|DiffRest]$ **in**
for_all_diff($DiffRest$,
new_diff(synchron($Connect, DiffFirst, GR[DiffFirst],$
$AS[DiffFirst], HHAs_{i+1}$),
$DiffFirst, NewDiff, GR, Cond, AS, Assign$),
$Connect, HHAs_{i+1}$);

define function new_diff($HHAsConnect, HHA, HHAsDiff, GR, Cond, AS, Assign$)
  if $HHAsConnect = \emptyset$ **then**
    $(HHAsDiff, GR, Cond, AS, Assign)$
  else
    let $HHAsConnect = [\langle HHAsend, \langle Receive, SyCond\rangle, \langle Send, SyAss\rangle, HHArecv\rangle | HHAsRest]$ **in**
      if $HHAsend = HHA$ **then**
        if $HHArecv = HHAenv$ **then**
          new_diff($HHAsRest, HHA, HHAsDiff,$
                   $GR, Cond \wedge SyCond,$
                   $AS, Assign \cup \{SyAss\}$)
        else
          let $\langle l, \langle TrGR, TrCond\rangle, \langle TrAS, TrAssign\rangle, l'\rangle$
              send_tuple($Send$,fdelta(flat($HHArecv$))) **and**
              $GR[HHArecv] = TrGR$ **and** $AS[HHArecv] = TrAS$ **in**
            new_diff($HHAsRest, HHA, HHAsDiff \cup \{HHArecv\},$
                     $GR + GR[HHArecv], Cond \wedge SyCond \wedge TrCond,$
                     $AS + AS[HHArecv], Assign \cup \{SyAss\} \cup \{TrAssign\}$)
      else
        if $HHAsend = HHAenv$ **then**
          new_diff($HHAsRest, HHA, HHAsDiff,$
                   $GR, Cond \wedge SyCond,$
                   $AS, Assign \cup \{SyAss\}$)
        else
          let $\langle l, \langle TrGR, TrCond\rangle, \langle TrAS, TrAssign\rangle, l'\rangle$
              recv_tuple($Receive$,fdelta(flat($HHAsend$))) **and**
              $GR[HHAsend] = TrGR$ **and** $AS[HHAsend] = TrAS$ **in**
            new_diff($HHAsRest, HHA, HHAsDiff \cup \{HHAsend\},$
                     $GR + GR[HHAsend], Cond \wedge SyCond \wedge TrCond,$
                     $AS + AS[HHAsend], Assign \cup \{SyAss\} \cup \{TrAssign\}$);

define function synchron($Connect, HHA, GR[HHA], AS[HHA], HHAs_{i+1}$)
  if $Connect \quad \emptyset$ **then**
    $\emptyset$
  else
    let $Connect$
        $[\langle HHAsend, \langle Receive, SyCond\rangle, \langle Send, SyAss\rangle, HHArecv\rangle |$
         $ConnectRest]$ **in**
        if $(HHAsend = HHA) \wedge (HHArecv \notin HHAs_{i+1}) \wedge$
          $(Receive \subseteq AS[HHA])$ **then**
          $[\langle HHAsend, \langle Receive, SyCond\rangle, \langle Send, SyAss\rangle, HHArecv\rangle |$
           synchron($ConnectRest, HHA, GR[HHA], AS[HHA], HHAs_{i+1}$)]

else
if ($HHArecv = HHA$) $\wedge$ ($HHAsend \notin HHAs_{i+1}$) $\wedge$
($Send \subseteq GR[HHA]$ **then**
[$\langle HHAsend, \langle Receive, SyCond\rangle, \langle Send, SyAss\rangle, HHArecv\rangle|$
synchron($ConnectRest, HHA, GR[HHA], AS[HHA], HHAs_{i+1}$)]
else
synchron($ConnectRest, HHA, GR[HHA], AS[HHA], HHAs_{i+1}$);

define function send_tuple($Send$,[$\langle l, \langle GR, Cond\rangle, \langle AS, Assign\rangle, l'\rangle|FdeltaRest$])
if $Send \subseteq GR$ **then**
$\langle l, \langle GR, Cond\rangle, \langle AS, Assign\rangle, l'\rangle$;
else send_tuple($Send$,$FdeltaRest$);

define function recv_tuple($Receive$,[$\langle l, \langle GR, Cond\rangle, \langle AS, Assign\rangle, l'\rangle|FdeltaRest$])
if $Receive \subseteq AS$ **then**
$\langle l, \langle GR, Cond\rangle, \langle AS, Assign\rangle, l'\rangle$;
else recv_tuple($Receive$,$FdeltaRest$);

Anhang D

Grammatik von MODEL-HS

```
BNF zu Hybriden Systemen in MODEL-HS:
=====================================

<HybridSystem> ::= system <Identifier> [<FormalParam>] ;
                     [<Import>]
                     [<Declaration>]
                     [<Synchronisation>]
                   endsystem <Identifier>.
_____________________________________________________________
<FormalParam> ::= <ParenthOpen> <ParamIdent> <ParenthClose>

<ParenthOpen> ::= (

<ParenthClose> ::= )

% in statischer Semantik beschreiben, dass wenigstens Signale
% vorhanden sein muessen, das heisst hier alle Moeglichkeiten,
% offen lassen in stat. Semantik Design,
% Restriktionen beschreiben

<ParamIdent> ::= <Invariant>
                 <Activities>
                 [<Parameters>]
                 [<Clocks>]
                 [<Variables>]
                 [<Signals>]

<Invariant> ::= invariant <Identifier> ;
```

```
<Activities> ::= activity <IdentList> ;

<Parameters> ::= parameter <IdentList> ;

% Bezeichner koennen als Array auftreten, d.h.unter gleicher,
% Benennung zugeordnet zu unterschiedlichen Bloecken bzw.
% Prozessen, so laesst sich die Anzahl beliebig variieren

<IdentList> ::= <IdentArray> [, <IdentList>]

<IdentArray> ::= <Identifier> [<Array>]

<Array> ::= <BracketOpen> <Number> <BracketClose>

<BracketOpen> ::= [

<Number> ::= <Digit> [<Number>]

<BracketClose> ::= ]

<Clocks> ::= clock <IdentList> ;

% in statischer Semantik, dass wenigstens eins vorhanden ist,
% abpruefen

<Variables> ::= variable
                  [<DiscreteVars>]
                  [<ContinuousVars>]
                  [<ControlVars>]

<DiscreteVars> ::= discrete <IdentList> ;

<ContinuousVars> ::= continuous <IdentList> ;

% Variablen, die eingefuehrt wurden, um in Bloecken Automaten-,
% Block- und Signalauftreten gleicher Art zusammenfassend
% behandeln zu koennen und in Automaten zusaetzliche
% Transitionen und Zustaende zu sparen diese Variablen koennen
% im Block deklariert werden und an Automaten weitergereicht
% werden

<ControlVars> ::= control
                      [<InBlock>]
```

```
                    [<InAutomaton>]

<InBlock> ::= in_block <IdentList>;

<InAutomaton> ::= in_automaton <IdentList>;

% InSignals muessen vorhanden sein, in statischer Semantik
% ueberpruefen wenn erweiterter Bezeichner wie bei QualifyRest,
% dann Liste von Signalen moeglich
% vor Punkt erst Automat angeben, nach Punkt jeweilige
% Signale wann Semikolon gesetzt wird und wann nicht, in
% statischer Syntax beschreiben

% ACHTUNG: im Block sind synch und refine nicht relevant,
% erst bei formalen Parametern eines Automaten

<Signals> ::= signal
                [<InSignals>]
                [<OutSignals>]

% kein Semikolon, da dahinter entweder 'out' oder schliessende
% Klammer folgt

<InSignals> ::= in <QualifyList>

% kein Semikolon, da dahinter immer schliessende Klammer folgt

<Outsignals> ::= out <QualifyList>

<QualifyList> ::= <QualifySignals> [,<QualifyList>]

% erster Teil, um ganz normal ueber Punktnotation ein Signal
% oder Liste von Signalen weiterzureichen,
% zweiter Teil, um Verknuepfungsoperatoren fuer Kanten laut
% HyCharts zu realisieren mit Hilfe der Parameterliste, so
% koennen mehrere Signale auf genau ein Signal gelegt werden
% --> siehe auch ,nested words' von Alur

<QualifySignals> ::= <IdentArray> [<QualifyRest>] |
                     <BracketOpen> <QualifyList> <BracketClose>

% in statischer Syntax pruefen, ob wirklich gesamte Liste, nur
% dann schliessende und oeffnende Klammer,
```

```
% beide Klammern muessen vorhanden sein -> doch diese Dinge
% haben noch nichts in der Grammatik zu suchen

<QualifyRest> ::= . [<Open>] <IdentList> [<Close>]

<Open> ::=

<Close> ::=
```

```
<Import> ::= import from <Libraries>
                [<BlockImport>]
                [<AutomImport>]

<Libraries> ::= <SimpleIdentList>

<SimpleIdentList> ::= <Identifier> [, <SimpleIdentList>]

<BlockImport> ::= block_import
                     <ImportList>

<ImportList> ::= <Identifier> [<FormalParam>] ; [<ImportList>]

<Autom_import> ::= automaton_import
                      <AutoImportList>

<AutoImportList> ::=
         <Identifier> [<AutoFormalParam>] ; [<AutoImportList>]

<AutomFormalParam> ::=
         <ParenthOpen> <AutoParamIdent> <ParenthClose>

<ParenthOpen> ::= (

<ParenthClose> ::= )

% in statischer Semantik beschreiben, dass wenigstens Signale
% vorhanden sein muessen, das heisst hier alle Moeglichkeiten
% offen lassen, in stat. Semantik Design, Restriktionen
% beschreiben

<AutoParamIdent> ::= [<Invariant>]
                     [<Activities>]
```

```
                    [<Parameters>]
                    [<Clocks>]
                    [<Variables>]
                    [<AutoSignals>]

<AutoSignals> ::= signal
                  [<SynchronSignals>]
                  [<RefineSignals>]

<SynchronSignals> ::= synchron
                        [<InSignals>]
                        [<OutSignals>]

<RefineSignals> ::= refine
                      [<InSignals>]
                      [<OutSignals>]
```

```
% eines muss zumindest auftreten oder sogar in Abhaengigkeit
% voneinander

<Declaration> ::= declaration
                  [<BlockDecl>]
                  [<AutomatonDecl>]
                  [<Clocks>]
                  [<Variables>]
                  [<InternSignals>]

% Signale, die nicht nach aussen gesandt werden oder von der
% Umgebung kommen, sondern an innere Automaten weitergeleitet
% bzw. von diesen empfangen werden
% gibt kein send und receive, da Punktnotation von
% MODEL-HS nutzbar!!!

<InternSignals> ::= signal <QualifyList> ;

% wenigstens eines muss erscheinen

<BlockDecl> ::= blocks
                [<Instances>]
                [<Blocks>]

% Aritaet als Hinweis auf welches importierte Modul bei
```

```
% gleicher Bezeichnung zugegriffen wird

<Instances> ::=
         <IdentList> : <Identifier> [<Arity>] ; [<Instances>]

<Arity> ::= <ParenthOpen> <Number> <ParenthClose>

<Blocks> ::= <Block> [<Blocks>]

% leerer Block ist zugelassen

<Block> ::= <Identifier> [<FormalParam>] ;
              [<Import>]
              [<Declaration>]
              [<Synchronisation>]
            endblock <Identifier> ;

<Synchronisation> ::= synchronisation
                        [<InitConnect>]
                        [<BlockDesign>]

<InitConnect> ::= initialisation
                        [Conditions]
                        [Assignments]

% 'or' zur Zeit noch nicht integriert, erst einmal solche
% Transitionen auseinandernehmen

<Conditions> ::= <Condition> [,<Conditions>]

<Condition> ::= <ArithmeticTerm> <Compare>

% Standardprioritaeten voraussetzen

<ArithmeticTerm> ::=
                  <MonadTerm> | <DyadTerm> |
                  <ParenthOpen> <ArithmeticTerm> <ParenthClose>

<MonadTerm> ::= <Literal> | <MonadOperator> <MonadTerm>

% Natural als Zahl erklaeren

<Literal> ::= <Identifier> | <Natural>
```

```
<MonadOperator> ::= ++ | --

<DyadTerm> ::= <Priority1> <PriorRest1>

<Priority1> ::= <Priority2> <PriorRest2>

<Priority2> ::=
       <Literal> | <ParenthOpen> <ArithmeticTerm> <ParenthClose>

<PriorRest1> ::= <DyadOperator1> <Priority1>

<PriorRest2> ::= <DyadOperator2> <Priority2>

<DyadOperator1> ::= + | -

<DyadOperator2> ::= * | /

<Compare> ::= <> <ArithmeticTerm> | = <ArithmeticTerm> |
              <= <ArithmeticTerm> | >= <ArithmeticTerm> |
              < <ArithmeticTerm> | > <ArithmeticTerm>

<Assignments> ::= <Assignment> [, <Assignments>]

<Assignment> ::= <Identifier> := <ArithmeticTerm>

<BlockDesign> ::= connection
                  <ForExpress> | <Express> [<ExpressRest>]

<ForExpress> ::= for <Assignment> to <Number> [step <StepWidth>]
                   <Express> [<ExpressRest>]
                 end_for;

<Express> ::=
             <Exchange> | <ParenthOpen> <Express> <ParenthClose>

<Exchange> ::= <Identifier> <ActualParam>

<ExpressRest> ::= <Op> <Express>

% || Parallelitaet = voellig unabhaengig nebeneinander
% laufende Prozesse,
% --> Sequenz = vollstaendig nacheinander ablaufend,
```

```
% v Alternative = ein Prozess oder der andere Prozess zu
% einer Zeit
% | Nebenlaeufigkeit = Parallelitaet mit staendiger
% Wechselwirkung zw. Prozessen, teilweise auf Signalebene
% parallel, jedoch nicht auf Prozessebene

<Op> ::= || | --> | v | '|'

% in statischer Semantik beschreiben, dass hier Luecken
% (leere Parameter) auftreten koennen und mindestens eine
% Gruppe vorhanden sein muss

% Condition fuer aktuelle Invarianten oder in Verbindung mit
% alternativen Signalen fuer Signale in Verbindung mit
% logischen Verknuepfungen
% Menge von Assignments fuer aktuelle Aktivitaeten
% Liste von arithmetischen Termen fuer Parameter
% QualifyList fuer Signale
% SignalCAList fuer Signale mit Bedingungen und Zuweisungen

<ActualParam> ::= <ParenthOpen> [<Condition> ,]
                                [<Assignments>,]
                                [<ArithmeticTermList>,]
                                [<QualifyList>,]
                                [<SignalCAList>]
                  <ParenthClose>

<ArithmeticTermList> ::=
                      <ArithmeticTerm> [, <ArithmeticTermList>]

% Signale verbunden mit Bedingungen oder Zuweisungen an
% Synchronisationsverbindung

<SignalCAList> ::= <SignalCondAss> [,<SignalCAList>]

<SignalCondAss> ::= <IdentArray> [<CondAssRest>]

% in statischer Syntax pruefen, ob wirklich gesamte Liste, nur
% dann schliessende und oeffnende Klammer,
% beide Klammern muessen vorhanden sein -> doch diese Dinge
% haben noch nichts in der Grammatik zu suchen

<CondAssRest> ::= . [<Open>] <IdentCAList> [<Close>]
```

```
<IdentCAList> ::= <IdentCondAss> [, <IdentCAList>]

<IdentCondAss> ::=
          <Ident> <ParenthOpen> [<Conditions>]
                                [<Assignments>] <ParenthClose>

% wenigstens eines

<AutomatonDecl> ::= automata
                       [<Instances>]
                       [<Automata>]

<Automata> ::= <Automaton> [<Automata>]

<Automaton> ::= <Identifier> [<AutomFormalParam>] ;
                  [<Refinement>]
                  <DeclAutom>
                  <Behaviour>
                endautomaton <Identifier> ;

<Refinement> ::= refinement from <Libraries> by <ImportList>

% Clocks und Lokationen muessen vorhanden sein, in statischer
% Semantik formulieren

<DeclAutom> ::= declaration
                  [<Clocks>]
                  [<Variables>]
                  [<Locations>]

<Locations> ::= location
                  [<SimpleLocDecl>]
                  [<ComplexLocDecl>]

<SimpleLocDecl> ::= simple <SimpleLocations>

<SimpleLocations> ::= <IdentList>;

<ComplexLocDecl> ::= complex
                       [<Instances>]
                       [<ComplexLocations>]
```

```
<ComplexLocations> ::= [<Automata>]
                       [<Blocks>]


<Behaviour> ::= behaviour
                    [<ParamMatching>]
                     <Initialisation>
                     <Termination>
                     <Transitions>

% Die Verknuepfung formalen Parameter der Automaten mit
% aktuellen Parametern erfolgt im<AutomatonBody> im Bereich
% <ParamMatching>

<ParamMatching> ::= matching
                      <Matching>

<Matching> ::= <Match> ; [<Matching>]

% Alle Instanzen anderer Automanten, die Parameter besitzen
% und in diesem Automaten instanziiert wurden muessen hier
% aufgefuehrt werden, und ihre formalen Parameter mit aktuellen
% Parameter gematcht werden.
% Dies gilt auch fuer Automaten, die in diesem Automaten
% deklariert wurden.

<Match> ::= <Identifier> <ActualParam>


% Anfangslokation muss vorhanden sein

<Initialisation> ::= initialisation
                       if <DiscreteConditionList>
                       <InitialLocation>

<DiscreteConditionList> := <DiscreteCondition>
                           [or <DiscreteConditionList>]

% Action kann leer sein

<DisreteCondition> := <Guard> [-> <Action>]

% an Guard zu sehen, dass bei uns alles mit gesendeten oder
```

```
% empfangen Signalen zusammenhaengt, auch kein interenes
% Ereignis ohne Signal da wir Woerter angeben wollen
% true ist ein Wert, der nur fuer eine Initialtransition
% auftreten kann, um Werte einfach zu setzen
% ! unbedingt in Signallisten noch optionale Signale einfuegen!

<Guard> ::=  true |
             <Receive> [,<Conditions>] | <Conditions>

<Receive> ::= receive <Open> <SignalList> <Close>

% <Assignments> koennen auch leer sein

<Action> ::=  <Send> [,<Assignments>] | <Assignments>

<Send> ::= send <Open> <SignalList> <Close>

% steht fuer Konjunktion ACHTUNG: Hier noch Beschreibung fuer
% regulaere Ausdruecke einfuegen

<SignalList> ::= <IdentArray> [, <SignalList>]

<InitialLocation> ::= location <Identifier> ;

<Termination> ::= termination <Identifier> [<Iden_list>] ;

<Transitions> ::= transition
                  <ContinuousTransitions>
                  <DiscreteTransitions>

<ContinuousTransitions> ::= continuous <ContinuousTrans> ;

<ContinuousTrans> ::=  <Continuous> [; <ContinuousTrans>]

% Invarianten unterteilt in Bedingungen, die fuer mehrere
% Zustaende und Aktionen gelten, sind als gemeinsame
% Bedingungen unter while aufgefuehrt und in Bedingungen, die
% speziellen Zustaenden zugeordnet werden, sind diesen
% Zustaenden in Kombination mitgegeben

<Continuous> ::= [while <CommonInvariants>]
                        <DoInList>
```

```
<CommonInvariants> ::= <Condition> [, <Conditions>]

<InDoList> ::= [do <Assignment> [, <Assignments>]]
               in <IdentCombine> [ <IdentCombList>]
               [DoInList]

<IdentCombList> ::= <IdentCombine> [ <IdentCombList>]

<IdentCombine>  ::= <Identifier> [<ParenthOpen>
                                    [while <Conditions>]
                                    [do <Assignments>]
                                  <ParenthClose>] ;

<DiscreteTransitions> ::= discrete <DiscreteTranses> ;

<DiscreteTranses> ::= <DiscreteTrans> [; <DiscreteTranses>]


<DiscreteTrans> ::= [if <DiscreteConditionList>]
                     for
                      <OpenBrace>
                         <Discrete> [; <DiscreteList>]
                      <CloseBrace>
<OpenBrace> ::= {

<CloseBrace> ::= }

<DiscreteList> ::= <Discrete> [; <DiscreteList>]

% mehrere Lokationen koennen von der gleichen Lokation unter
% unterschiedlichen Bedingungen und vielleicht unterschiedlichen
% Aktionen erreicht werden bzw. mehrere Lokationen koennen zu
% einer Lokation ueber unterschiedliche Bedingungen oder auch
% unterschiedliche Aktionen fuehren

% in statischer Sematik ist spaeter noch zu klaeren , dass nicht
% verlassende und erreichbare Lokation mit dem Guard und der
% Action verbunden sind, nur eine Lokation von beiden

<Discrete> ::=
            ( leave <SimpleIdentList> reach <SimpleIdentList> |
              remain <SimpleIdentList> )
              [if <DiscreteConditionList>]
```

```
Typen in der Bibliothek:
========================

<InLibrary> ::= library <Identifier>
                  [automata <Automata>]
                  [blocks <Blocks>]
                end <Identifier>.
```

Anhang E

Beispiele

E.1 Studienbüro als Mehrfachprozess

Um eine Bearbeitung mehrere Prüfungsabläufe in einem Studienbüro aus dem Abschnitt 4.4.1 zu ermöglichen, können die zugrundeliegenden Automaten wie in der Abbildung E.1 abgewandelt werden.

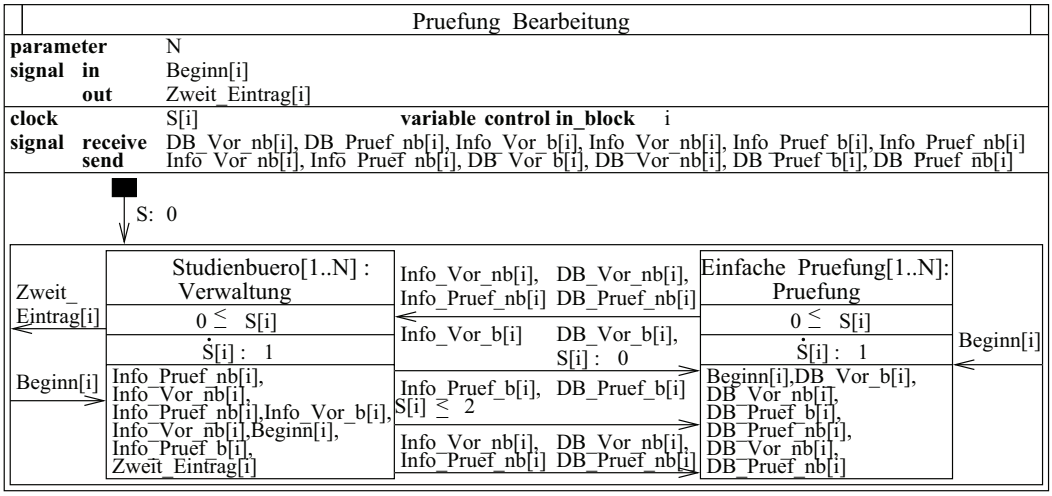


Abbildung E.1: Produktautomat der Synchronisation

In unseren Sprachen besteht die Möglichkeit, Instanznamen mit einer Bereichsbeschreibung zu verknüpfen. Die Bereichsangabe gibt die möglich Anzahl an Inkarnationen, die zur Laufzeit von einem Typ geschaffen werden können, an. Laut der Abbildung E.1 kann das Studienbüro für ein bis 'N' Verwaltungsprozesse instantiiert werden und der Prozess der 'Einfachen_Pruefung' mindestens für einen Studenten und maximal für 'N' Studenten. Der Parameter 'N' wird von der Umgebung festgelegt. Die Variable 'i' repräsentiert

für die gegebenen Signale die konkrete Instanz aus dem Bereich ’1..N’, von der die Signale gesendet oder empfangen werden.

E.2 Beispiel der symbolischen Simulation

Ein weiteres Beispiel zur symbolischen Simulation wurde erarbeitet, um den Einfluss zusätzlicher Bedingungen durch einen Nutzer der Simulation darzustellen. Ein Nutzer verlangt zusätzlich zu den gegebenen Bedingungen des Beispiels aus dem Abschnitt 5.3.5, dass die Voraussetzung innerhalb der ersten 19 Zeiteinheiten (Monate) abzulegen ist.

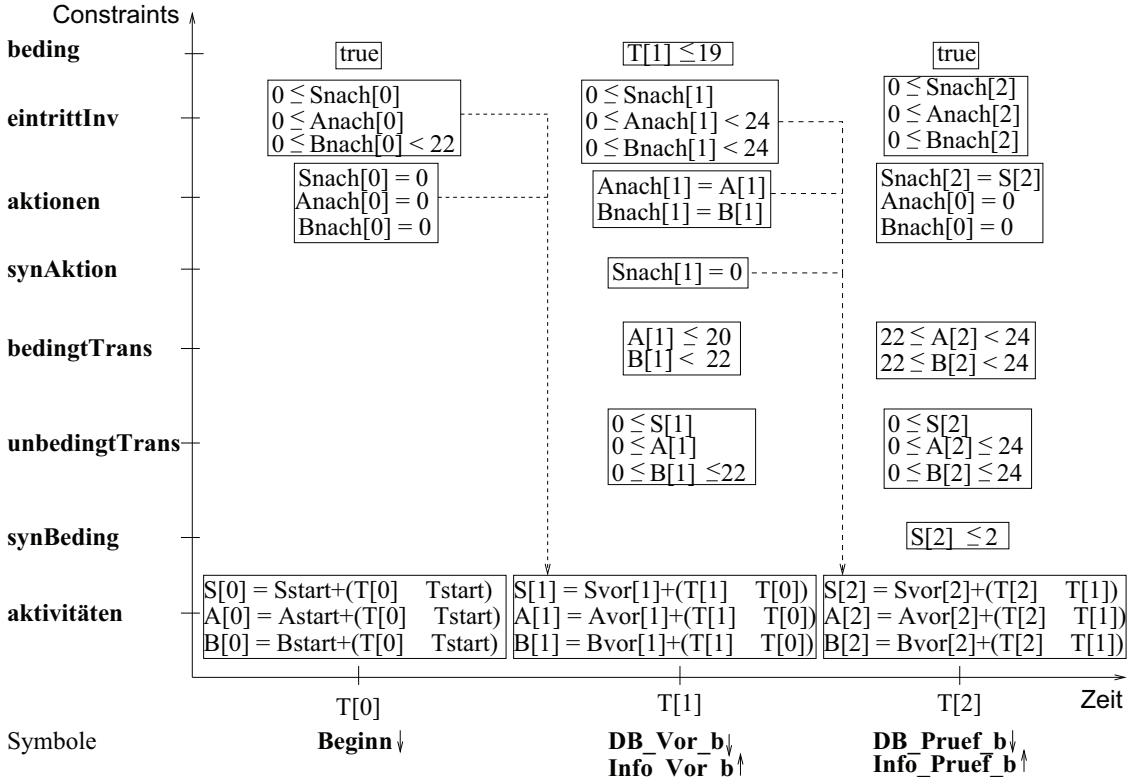


Abbildung E.2: Lauf mit Nutzerbedingung für T[1]

Aus dieser Bedingung ergibt sich, wie in der Abbildung E.3 zu sehen ist, im dritten Aufruf in Zusammenhang mit den Bedingungen ’T[2]-T[1] $\leq$ 2’ und ’22 $\leq$ T[2] < 24’ ein ***Widerspruch***. Entweder können zur Erfüllung dieser Forderung die Bedingungen die Prüfungszeiträume zwischen 22 und 24 Zeiteinheiten nicht eingehalten werden oder die Bedingung der übergeordneten Verordnung, dass eine Prüfung maximal 2 Zeiteinheiten nach dem Bestehen der Voraussetzung absolviert werden muss. Somit sind entweder die Ansprüche des Nutzers zu ändern oder das Modell ist zu verbessern.

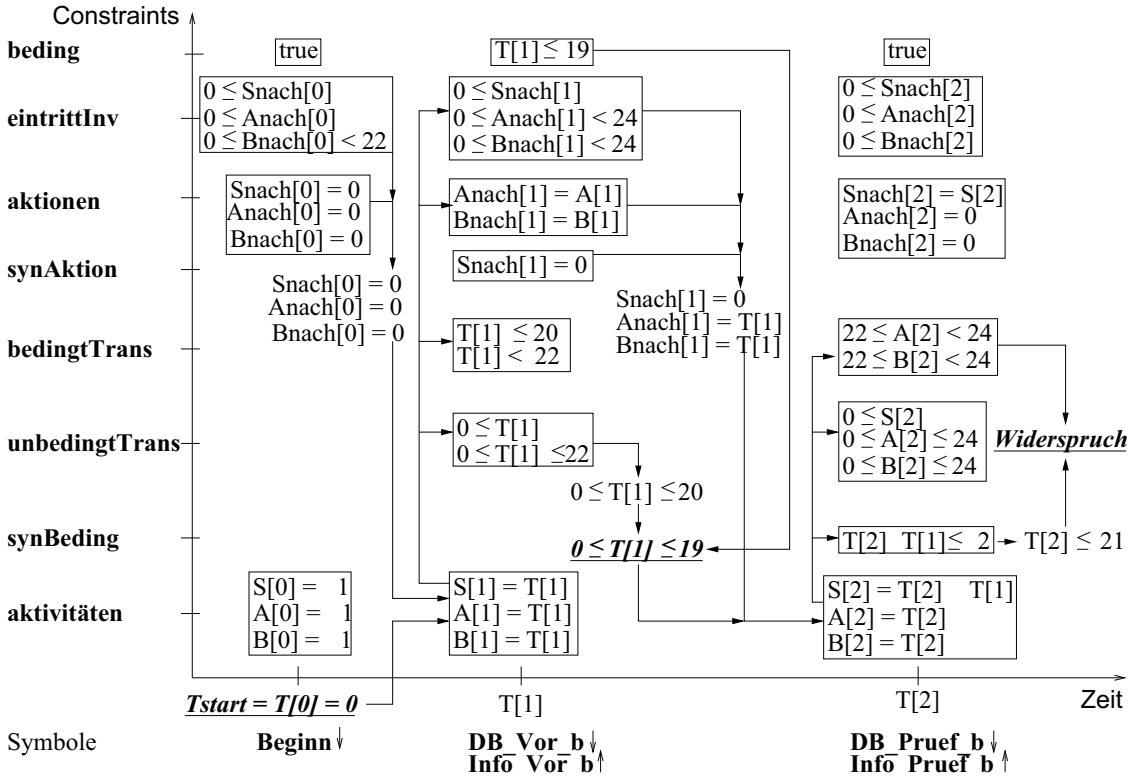


Abbildung E.3: Ergebnisse des Laufes

E.3 Vollständiges Beispiel

Dieser Abschnitt verdeutlicht die Vorgehensweise der Verfeinerung und Synchronisation von Automaten praktisch anhand eines vollständigen Beispiels. Hier werden sowohl die Hierarchie, als auch das Produkt der Automaten und die Ausführung geschachtelter und synchronisierter Läufe untersucht. Synchronisierende und hierarchische hybride Automaten werden dabei in flache Automaten überführt. Theoretische Erkenntnisse zum Model Checking hierarchischer Strukturen, die vorher in flache Strukturen überführt wurden, sind in [Loh05] gegeben. Demgegenüber sind in [AY01] theoretische Grundlagen zur Ausführung von Läufen in hierarchischen Zustandsmaschinen erarbeitet worden. Die Ausführung auf der Basis geschachtelter Wörter regulärer Sprachen, die die Läufe darstellen, wurde in [AM06, ACM06, AAB+07] untersucht. In synchronisierend und hierarchisch hybriden Automaten entstehen synchronisierte und geschachtelte Zeitwörter, die die Grundlage der Ausführung der symbolischen Simulation bilden. Anhand solcher Zeitwörter können die Auswirkungen der Synchronisation und Verfeinerung von Automaten auf deren Verhalten untersucht werden. Zur Darstellung der Semantik werden die nebenläufigen und geschachtelten Zeitwörter in rein sequentielle symbolische Zeitwörter, die die Basis für die Ausführung von Läufen in CLP bilden, überführt.

E.3.1 Beschreibung des Beispiels

Anhand eines Beispiels des Reinigens, Erwärmens und Verformens eines Bambusstabes mit einer automatischen Steuerung wird die Überführung des synchronisierend und der hierarchisch hybriden Automaten in flache Automaten gezeigt. Im Gegensatz zu unseren Studiensystemen wurde hier ein technisches Problem gewählt, da in Studiensystemen mit normalen Studienverläufen keine Zyklen auftreten. Zyklen bilden wesentliche Merkmale der bereits gut untersuchten technischen Systeme, welche eine spezielle Behandlung in der Beschreibung als auch bei der Transformation in einen flachen Automaten verlangen. Wie in Abbildung E.4 ist der Vorgang der Reinigung als eigenständig wiederverwendbarer Prozess den Vorgängen des Vorheizens und Biegens eines Bambusstabes vorgelagert.

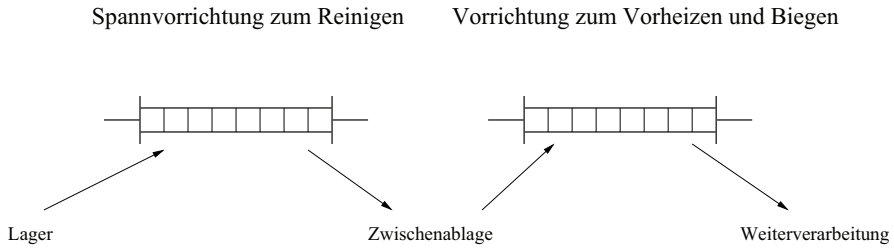


Abbildung E.4: 'Reinigen' gefolgt vom 'Vorheizen und Biegen' eines Bambusstabes

Die Vorgänge des Vorheizens und Biegens sind eng miteinander verbunden. Ein Vorheizen ohne nachfolgendes Biegen soll nicht möglich sein, weshalb beide Vorgänge in einem gemeinsamen Prozess beschrieben sind. Wie die Abbildung E.5 zeigt, ist in eine Spannvorrichtung ein Bambusstab einzulegen, um welchen in gleichen Abständen Wärmequellen angebracht sind. Die Wärmequellen werden intervallmäßig an- und abgeschaltet, so

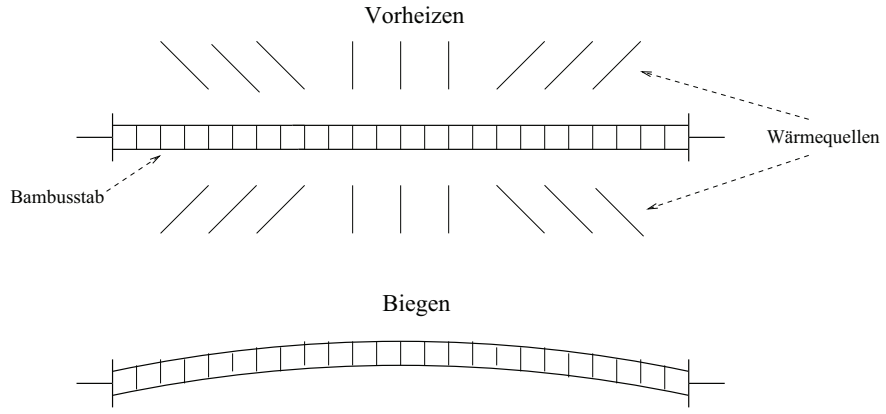


Abbildung E.5: Vorheizen und Biegen eines Bambusstabes

dass der Bambusstab entsprechend seines Durchmessers D langsam bis in den inneren

Kern auf eine vorgegebene Temperatur T erwärmt werden kann, ohne dabei an den äußeren Schichten zu verbrennen. Die Wärmequellen sind zusätzlich mit einer Sicherung ausgestattet worden, welche im folgenden Text zu einer weiteren Verfeinerung des Erwärmungsverlaufes führt. Hat der Bambusstab die erforderliche Temperatur erreicht, so wird durch den Vorgang 'Biegen' die Form des Stabes verändert. Nach Beendigung der Verformung wird der Stab aus der Spannvorrichtung entfernt und der Kreislauf wird durch das Einlegen eines neuen Bambusstabes geschlossen.
Zur genauen Beschreibung des Verhaltens in Form von Wörtern wird das hybride System des gesamten Vorgangs von unten nach oben aufgebaut. Auf diese Art und Weise können synchronisierte und geschachtelte Wörter unmittelbar im Zusammenhang mit den gerade behandelten Abstraktionsebenen angegeben werden.

E.3.2 Automat 'Reinigen'

Zur Beschreibung des Reinigungsprozesses ist ein hierarchisch hybrider Automat modelliert, der keine weitere Verfeinerung aufweist. Nach der Aktivierung des Vorganges wird ein Bambusstab in die Spannvorrichtung eingelegt und danach für eine bestimmte Zeitspanne gesäubert. Ist der Stab vollständig gereinigt, so wird der Stab herausgenommen und ein neuer Bambusstab für einen nächsten Zyklus eingelegt.

Notation in MODEL-HS und VYSMO

Der Automatentyp der 'Reinigung' liegt in einer Bibliothek vor. In MODEL-HS wird der Typ in folgender Notation wiedergegeben:

```
Reinigung (invariant ReinInv;
           activity ReinAct DM;
           parameter DM;
           signal synchron in   eingelegt
                          out   sauber
                  refine  in    initr
                          out   neu_einlegen);
declaration
    clock X;
    location
        simple Bambusstab einlegen, Bambusstab reinigen, Bambusstab herausnehmen;

behaviour
    initialisation
        if receive{initr} -> X :  0;
        location Bambusstab einlegen;
    termination
```

```
        Bambusstab einlegen;
    transition
        continuous
            while 0 ≤ X
            do X :  X+1
            in Bambusstab einlegen, Bambusstab reinigen(while X ≤ 10),
                Bambusstab herausnehmen;
        discrete
            if true -> X :  0
            for
                {
                leave Bambusstab einlegen reach Bambusstab reinigen
                if receive{eingelegt}, 4 < X < 6;
                leave Bambusstab reinigen reach Bambusstab herausnehmen
                if 2*DM ≤ X -> send{sauber};
                leave Bambusstab herausnehmen reach Bambusstab einlegen
                if 2 ≤ X -> send{neu_einlegen};
                }
endautomaton Reinigung;
```

In VYSMO wird die Beschreibung wie in der Abbildung E.6 dargelegt. Die Reinigung

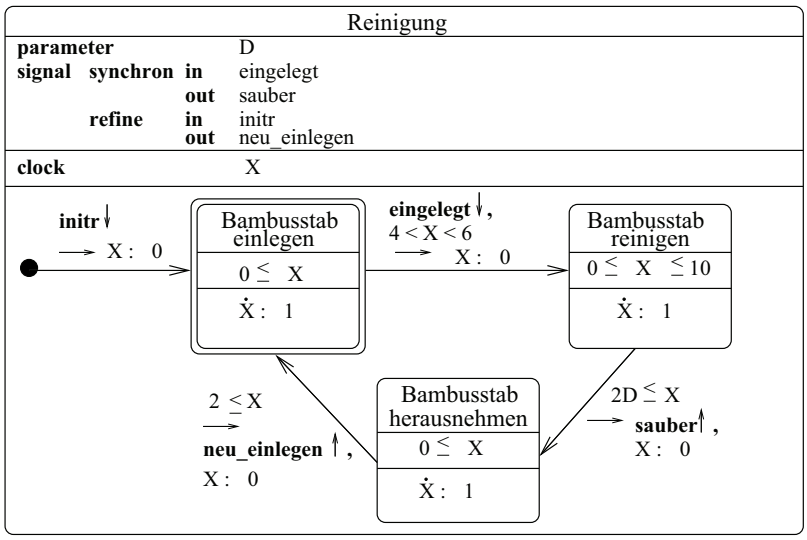


Abbildung E.6: ’Reinigung’ eines Bambusstabes als HHA

ist durch die drei aufeinanderfolgenden, zeitverbrauchenden Vorgänge ’Bambusstab einlegen’, ’Bambusstab reinigen’ und ’Bambusstab herausnehmen’ gekennzeichnet, die in einem unendlich langen Zyklus durchlaufen werden können. Ist ein Bambusstab richtig

eingelegt worden, so löst die Umgebung zwischen ’4’ und ’6’ Zeiteinheiten mit einem Signal ’eingelegt’ den Vorgang ’Bambusstab reinigen’ aus. Ist der Bambusstab in Abhängigkeit seines Durchmessers ’D’ vollständig gereinigt worden, so führt das Auslösen des Signals ’sauber’ zum Vorgang ’Bambusstab herausnehmen’. Nach der erfolgreichen Entfernung des Bambusstabes führt die Transition mit dem Signal ’neu_einlegen’ zur Endlokation ’Bambus einlegen’, die gleichzeitig die Startlokation eines weiteren Durchlaufes (Zyklus) ist.

Spezifizierte Wörter

Sämtliche Wörter, die das Verhalten des Automaten ’Reinigung’ bilden können, hängen von den Invarianten und Aktivitäten der Lokationen sowie den Bedingungen und Zuweisungen der Transitionen ab und werden entsprechend des Automaten der Abbildung E.6 wie folgt angegeben:

$\langle$ {**initr**}, T[0,0], {X},$\langle$release({}, ’true’), conclude({X’: 0}, $0 \leq$ X’)$\rangle\rangle$,
for i: 1 **to** n **do**
(
$\langle$ {**eingelegt**}, T[1,i], {X},$\langle$release({X: T[1,i]-T[0,i-1]}, $4 <$ X < 6),
conclude({X’: 0}, $0 \leq$ X ≤ 10)$\rangle\rangle$,
$\langle$ {**sauber**}, T[2,i], {X},$\langle$release({X: T[2,i]-T[1,i]}, $2D \leq$ X $\leq 10+\epsilon$),
conclude({X’: 0}, $0 \leq$ X’)$\rangle\rangle$,
$\langle$ {**neu_einlegen**}, T[0,i], {X},$\langle$release({X: T[0,i]-T[2,i]}, $2 \leq$ X),
conclude({X’: 0}, $0 \leq$ X)$\rangle\rangle$
)

n -> unendlich.
n kann 0 sein
nachher vereinfachen , indem Invarianten als urgent transitions und richtige Transitionsbedingungen zusammengefasst werden

E.3.3 Automat ’Heizen und Biegen’

Unter welchen Bedingungen der Kreislauf zum Vorheizen und Biegen eines Bambusstabes durchzuführen ist, wird mit dem HHA in Abbildung E.7 spezifiziert.

Notation in MODEL-HS und VYSMO

Der Automatentyp für den Prozess ’Heizen_und_Biegen’ entspricht der Notation folgender Module:

```
Heizen_und_Biegen (invariant HeizInv;
                   activity HeizAct;
                   parameter T, D;
                   signal synchron in   inithb, begin_biegen
                                  out   vorheiz, initi, anfangen, sichern, fertig
                                        entsichern, vorheiz_fertig, biegen_fertig
                          refine  in    neu_einlegen);
refined from Library by
    Intervall (invariant Inv;
               activity Act;
               parameter D;
               variable continuous Tj;
               signal synchron out initi, anfangen, sichern, entsichern
                      refine   in  weiter
                               out cycle(fertig));
declaration
    clock X;
    variable continuous Tj;
    location
       simple Bambusstab einlegen, Biegen, Bambusstab herausnehmen;
       complex Vorheizen: Intervall;

behaviour
    matching
        Vorheizen(0 ≤ X ∧ 15 ≤ Tj ≤ T+4, {X :  X+1, Tj :  Tj+1}, D, Tj, initi, anfangen,
                  sichern, entsichern, begin_biegen, ⟨vorheiz_fertig, fertig⟩);
    initialisation
        if receive{inithb} -> X :  0, Tj :  15;
        location Bambusstab einlegen;
    termination
        Bambussatb einlegen;
    transition
        continuous
            while 0 ≤ X
            do X :  X+1, Tj :  Tj+0
            in Bambusstab einlegen(while Tj   15), Biegen(while 15 ≤ Tj ≤ T+4),
               Bambusstab herausnehmen(while 15 ≤ Tj ≤ T+4);
        discrete
            if 4 < X < 6 -> send{vorheiz}
            for
              leave Bambusstab einlegen reach Vorheizen;
            if true -> X :  0
            for
```

{
leave Vorheizen **reach** Biegen
(**if** T+2 ≤ Tj **->** **send**{vorheiz_fertig} ∨
if receive{begin_biegen}, T ≤ Tj ≤ T+2);
leave Biegen **reach** Bambusstab herausnehmen
if 4 ≤ X **->** **send**{biegen_fertig};
leave Bambusstab herausnehmen **reach** Bambusstab einlegen
if receive{neu_einlegen}, 2 ≤ X **->** Tj : 15;
}
endautomaton Heizen_u_Biegen;

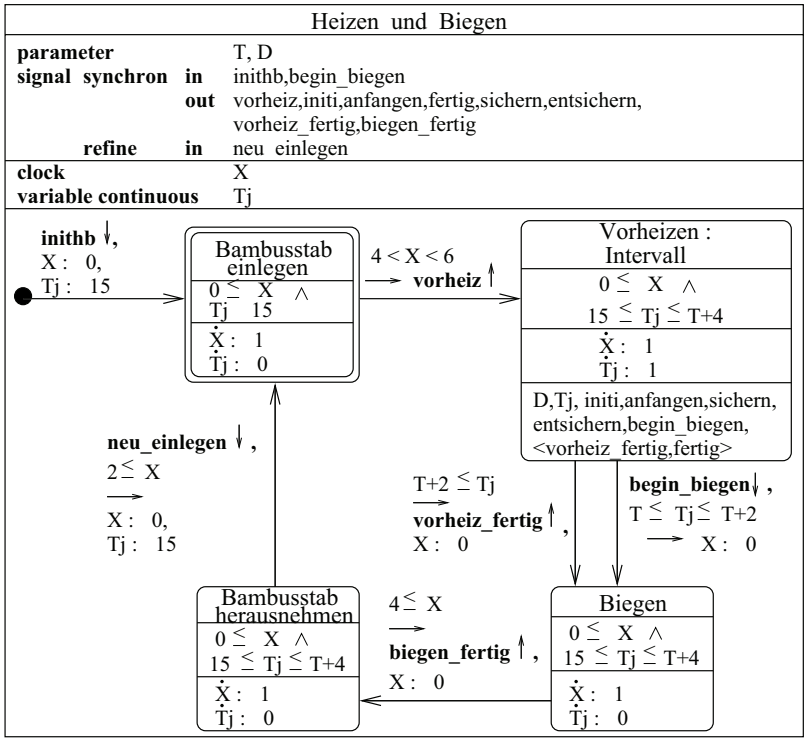


Abbildung E.7: Vorheizen und Biegen eines Bambusstabes als HHA

In dem beschriebenen hierarchisch hybriden Automaten zur Realisierung des Vorheizens und des Biegens werden nicht alle Bestandteile, die in der Definition 4.3.1 aufgeführt sind, verwendet. Der Automat besitzt eine lokale Uhr 'X' und eine kontinuierliche Variable 'Tj', die die gegenwärtige Temperatur des zu bearbeitenden Bambusstabes beinhaltet. Für die Lokation 'Vorheizen' ist durch die alternative Verwendung von automatisch bzw. manuell ausgelöster Transition eine Verzweigung im Kreislauf entstanden. Zum einen kann der Übergang von 'Vorheizen' zur Lokation 'Biegen' durch die automatisch ausgelöste Transition mit dem gesendeten Signal 'vorheiz_fertig' erfolgen. Zum anderen besteht die Möglichkeit, den Vorgang 'Vorheizen' durch die manuell ausgelöste Transition mit

dem Signal 'begin_biegen' zu verlassen. Hier wurde einem externen Nutzer des Systems erlaubt, innerhalb der Zeitspanne von T bis T+2 durch einen Knopfdruck den Zyklus des Vorheizens abzubrechen.

```
Intervall (invariant Inv;
           activity Act;
           parameter D;
           variable continuous Tj;
           signal synchron out initi, anfangen, sichern, entsichern
                  refine   in   weiter
                           out  cycle(fertig);
refined from Library by
    Sicherung (invariant SichInv;
               activity SichAct;
               parameter D;
               signal synchron out sichern, entsichern
                      refine   in   vor_entsichern
                               out  warten_ende);
declaration
    clock Y;
    variable discrete Tvor;
    location
       simple Heizen, Manuell erzeugter Vorgang;
       complex Warten: Sicherung;

behaviour
    matching
        Warten(0 ≤ YD ≤ 4D ∧ Tvor ≤ Tj ≤ Tvor+1, {Y :   Y+4, Tj :   Tj+0}, D, Tj,
               sichern, entsichern, weiter, anfangen);
    initialisation
        if true -> send{initi}, Y :   0, Tvor :   Tj;
        location Heizen;
    termination
        Warten, Manuell erzeugter Vorgang;
    transition
        continuous
            while 0 ≤ Y ∧ Tvor ≤ Tj ≤ Tvor+1
            do Y :   Y+0, Tj :   Tj+1
            in Heizen;
            while 1D ≤ Y
            do Y :   Y+0
            in Manuell erzeugter Vorgang;
        discrete
```

```
        for
            {
            leave Heizen reach Warten
            (if Tj-Tvor    1 -> send{fertig}
             leave Warten reach Heizen
             if 2D ≤ Y ≤ 3D -> send{anfangen}, Tvor :   Tj, Y :   0;
             leave Warten reach Manuell erzeugter Vorgang
             ifreceive{weiter}, 1D ≤ Y ≤ 3D;
             }
endautomaton Intervall;
```

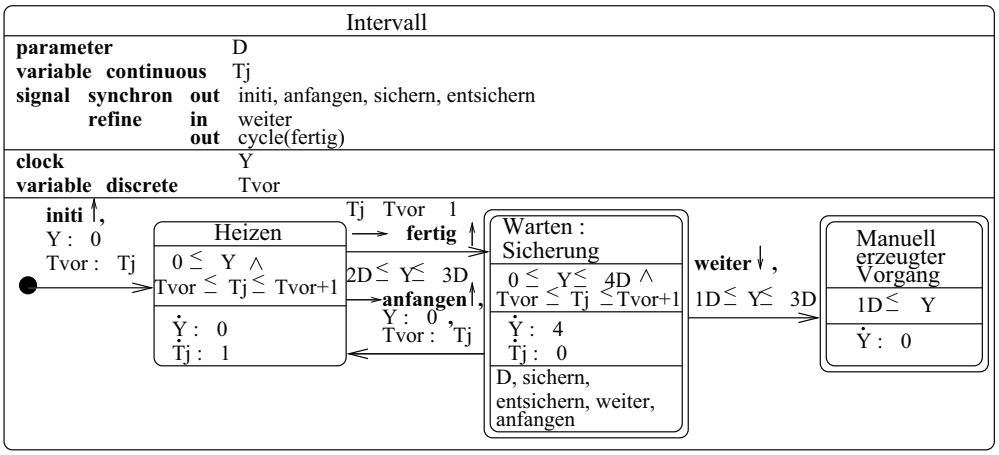


Abbildung E.8: Vorheizen als verfeinerte Abfolge

Die Lokation 'Vorheizen' ist eine komplexe Lokation, welche durch den in Abbildung E.8 beschriebenen Automaten 'Intervall' verfeinert wird. Da die Erwärmung des Bambusstabes nicht kontinuierlich erfolgen soll, sind Phasen erforderlich, in denen die Heizquellen Wärme abgeben als auch ausgeschaltet sind. Der hierarchisch hybride Automat 'Intervall' bietet mit den Lokationen 'Heizen' und 'Warten' die Möglichkeit, die beiden Phasen in einem Zyklus zu beschreiben. So können die Heizquellen entsprechend der Bedingungen für die Temperatur 'Tj-Tvor 1' und die Zeit '2D ≤ Y ≤ 3D' ständig an- und abgeschaltet werden. Dieser Zyklus kann durch das Eintreten des Ereignisses 'fertig' automatisch verlassen werden oder durch ein von einem externen Nutzer ausgelöstes Ereignis 'weiter', welches zusammen mit dem Verbleiben in der Lokation 'Warten' auftreten muss. Im unteren Teil der Lokation 'Vorheizen' sind die aktuellen Parameter und Signale, die mit den formalen Parametern und Signalen des Automaten 'Intervall' verbunden werden, aufgeführt. Signale, die ausschließlich der Synchronisation dienen, werden in die Schnittstellen kopiert, um zur Synchronisation mit hybriden Automaten aus der Umgebung zur Verfügung zu stehen. Wird keine Kopie in der Schnittstelle angelegt, so können die Signale nur für interne Synchronisationsvorgänge zur Verfügung stehen.

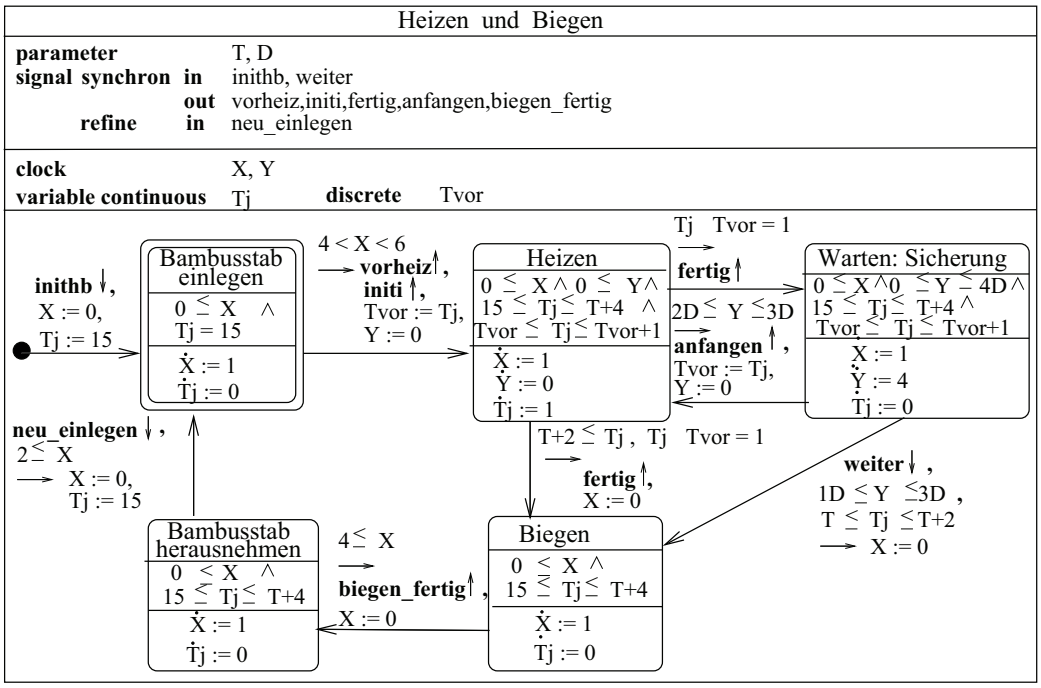


Abbildung E.9: Heizen und Biegen, 1. Verfeinerungsstufe

Sicherung (**parameter** D;
signal synchron out sichern, entsichern
entsichern, vorheiz_fertig, biegen_fertig
refine **in** vor_entsichern
out warten_ende);
refined from Library **by**
declaration
clock Z;
location
simple Warten auf Entsichern, Warten auf Heizen,
Manuell weiter, Automatisch weiter;

behaviour
initialisation
if true -> send{sichern}, Z : 0;
location Warten auf Entsichern;
termination
Manuell weiter, Automatisch weiter;
transition
continuous
while 0 ≤ Z

do Z : Z+4
in Warten auf Entsichern(Z $\leq$ 2D), Warten auf Heizen(Z $\leq$ 4D),
Manuell weiter, Automatisch weiter;
discrete
for
{
leave Warten auf Entsichern **reach** Warten auf Heizen
(**if** 1D $\leq$ Z **-> send**{entsichern};
leave Warten auf Entsichern**reach** Manuell weiter
if receive{vor_entsichern}, 1D $\leq$ Z $\leq$ 2D ;
leave Warten auf Heizen **reach** Automatisch weiter
if 2D $\leq$ Z $\leq$ 3D **-> send**{warten_ende};
}
endautomaton Sicherung;

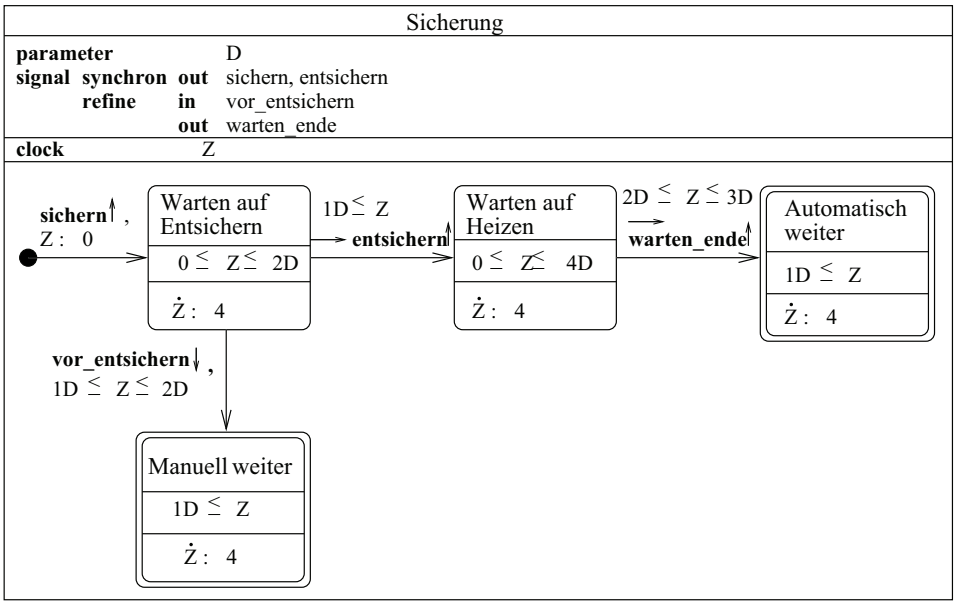


Abbildung E.10: Warten als verfeinerte Abfolge

In einer zweiten Verfeinerungsstufe wird die Lokation 'Warten' des hierarchisch hybriden Automaten 'Intervall' durch den Automaten 'Sicherung' der Abbildung E.10 detaillierter beschrieben. Die Heizquellen sind mit einer zusätzlichen Sicherung ausgestattet worden, die mit dem Abschalten der Heizquellen automatisch aktiviert wird. Der von dem externen Nutzer ausgelöste Knopfdruck zum Starten des Vorganges 'Biegen' kann nur dann akzeptiert werden, wenn die Sicherung der Heizquellen aktiv ist. Die manuelle Aktion 'vor_entsichern' kann somit nur in der Lokation 'Warten auf Entsichern' ausgeführt werden. Sind die Heizquellen nach einer bestimmten Zeiteinheit wieder entsichert, so kann

in Abhängigkeit von der Zeit der Entsicherung eine weitere Wartezeit auf den Heizvorgang in der Lokation 'Warten auf Heizen' entstehen. Hier kann der Leser sich vorstellen, dass hardwaretechnisch verschiedene Sicherungen und Heizquellen mit unterschiedlicher An- und Abschaltdauer eingesetzt werden sollen, deren optimales Zusammenwirken mit unserem Modell vor dem Einsatz überprüft werden kann. Durch die Zeit '$2D \leq Z \leq 3D$' wird festgelegt, in welchem Zeitraum der Vorgang des Heizens wieder automatisch einsetzt. Jeder Durchlauf, der mit dem Signal 'fertig' endet, ist ein akzeptierter Durchlauf im Sinne des Automaten 'Intervall'. Dieses Akzeptanzverhalten kann durch die Verfeinerung zeitlich eingeschränkt werden, soll jedoch nicht vollständig verloren gehen. In unserem Ansatz wird die Anfangslokation des hybriden Automaten, welcher die Lokation 'Warten' verfeinert, im erweiterten Kontext des Automaten 'Intervall' als Endlokation markiert.

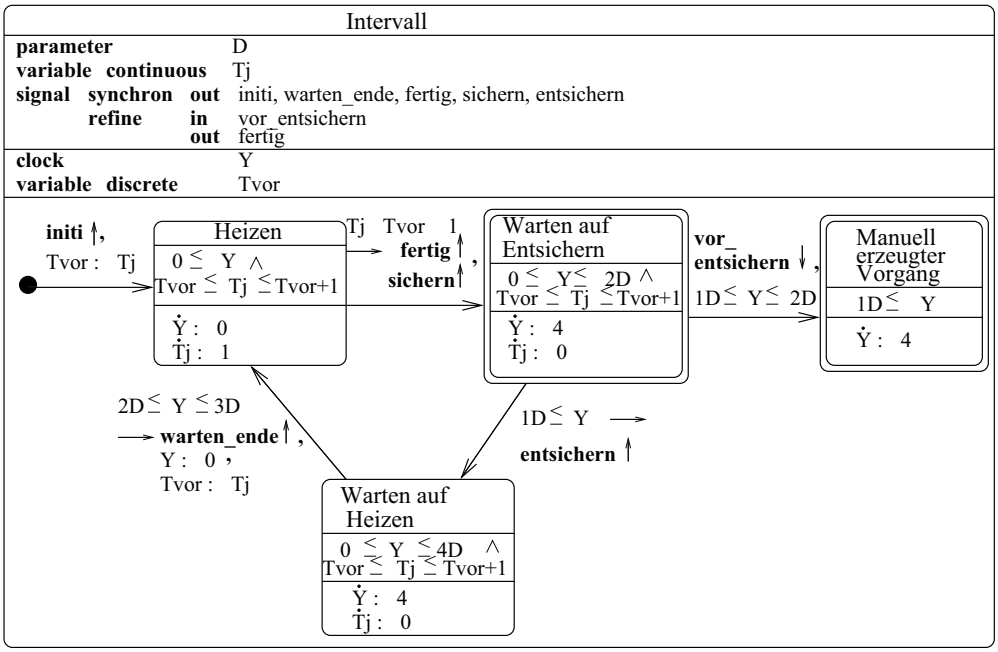


Abbildung E.11: Flacher Automat zum Vorheizen

In der Abbildung E.11 ist der Automat dargestellt, welcher infolge des Ersetzens der komplexen Lokation 'Warten' durch den Automaten 'Sicherung' entsteht. Hier können noch einmal anschaulich die Merkmale der in Kapitel 3 vorgestellten Verfeinerung hierarchisch hybrider Automaten dargelegt werden. Die komplexe Lokation 'Warten' stellt eine Endlokation innerhalb eines Zyklus dar, die zwei mögliche Folgelokationen 'Heizen' und 'Manuell erzeugter Vorgang' besitzt. Nach der Aufhebung der Hierarchie besteht der Zyklus neben der Lokation 'Heizen' aus einer akzeptierenden Lokation 'Warten auf Entsichern' und einer Lokation 'Warten auf Heizen'. Die Endlokationen 'Manuell weiter' und 'Automatisch weiter' des Automaten 'Sicherung' entfallen, da die Lokationen durch das Verhalten der Umgebung der komplexen Lokation 'Warten' im Automaten 'Intervall' überlagert werden. Somit folgt:

- der Lokation 'Warten auf Entsichern' die Lokation 'Manuell erzeugter Vorgang' anstelle der Lokation 'Manuell weiter' und
- der Lokation 'Warten auf Heizen' die Lokation 'Heizen' mit
  - darauffolgendem neuem Zyklus zu 'Warten auf Heizen' oder
  - nachfolgender Wartephase in 'Warten auf Entsichern' mit darauffolgend ausgelöster manueller Aktion 'vor_entsichern' zur Lokation 'Manuell erzeugter Vorgang'

  anstelle des Verhaltens der Lokation 'Automatisch weiter'.

Die der komplexen Lokation 'Warten' folgenden Transitionen mit den Signalen 'anfangen' und 'weiter' werden durch die Transitionen mit den Signalen 'warten_ende' und 'vor_entsichern' aus dem Automaten 'Sicherung' verfeinert. Im Fall der manuell ausgelösten Signale 'weiter' und 'vor_entsichern' erfolgt dabei eine zeitliche Einschränkung von '1D $\leq$ Y $\leq$ 3D' auf '1D $\leq$ Z $\leq$ 2D'. Da die lokalen Uhren 'Y' und 'Z' in den Aktivitäten mit denselben Taktraten fortschreiten, können sich diese beide Uhren gegenseitig ersetzen. Auf diese Art und Weise wird auch in der Abbildung E.11 des flachen Automaten nur die Uhr 'Y' mitgeführt. Die Invarianten der Lokationen 'Warten auf Entsichern' und 'Warten auf Heizen' im flachen Automaten entstehen durch die Konjunktion der Invariante der komplexen Lokation 'Warten' und den Invarianten der Lokationen 'Warten auf Entsichern' und 'Warten auf Heizen' des Automaten 'Sicherung'. Die Aktivitäten werden aus der Vereinigung der Aktivitäten der vorher genannten Lokationen erzeugt. Hierbei dürfen keine Inkonsistenzen innerhalb der Invarianten bezüglich der Aktivitäten auftreten. Diese Eigenschaft kann durch die symbolische Simulation nachgewiesen werden.
Wird die komplexe Lokation 'Vorheizen' in dem Automaten 'Heizen_und_Biegen' der Abbildung E.7 vollständig verfeinert, so entsteht der flache Automat in Abbildung E.12. Die Umwandlung zum flachen Automaten erfolgt ähnlich der vorherigen Beschreibung zur Beseitigung der Hierarchiestufe der komplexen Lokation 'Warten' des hierarchisch hybriden Automaten 'Intervall' mit zwei Unterschieden. Zum ersten entfallen sämtliche Markierungen als Endlokationen, die der verfeinernde Automat 'Intervall' aufgewiesen hatte, da die komplexe Lokation 'Vorheizen' des Automaten 'Heizen_und_Biegen' selbst keine Endlokation darstellt und somit zu keinem Zeitpunkt innerhalb dieser Lokation eine Trajektorie akzeptiert wird. Zum zweiten schreiten die Uhren 'X' der Lokation 'Vorheizen' und 'Y' des verfeinernden Automaten 'Intervall' mit unterschiedlichen Taktraten fort, so dass erst durch den Einsatz der symbolischen Simulation eine Vereinbarkeit der Invarianten unter diesen Taktraten festgestellt werden soll. Deshalb sind vorerst alle Invarianten und Aktivitäten rein syntaktisch zu vereinigen.

Spezifizierte Wörter

Beginnend mit dem Automaten für die Sicherung der Heizung werden die Wörter schrittweise von unten nach oben aufgebaut. Dabei wird gezeigt, wie sich über die Schachtelung

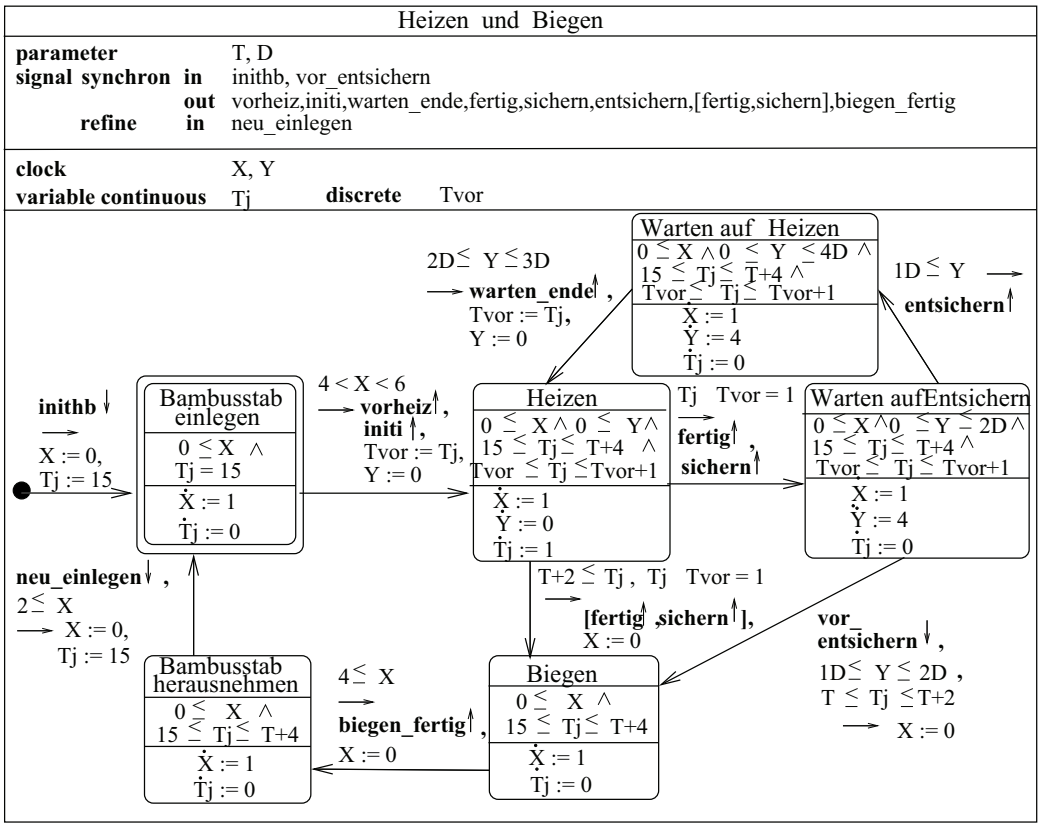


Abbildung E.12: Flacher Automat zum Heizen und Biegen

in Wörtern weitere detailliertere Wörter zur Verfeinerung erzeugen lassen.
Das Verhalten des Automaten 'Sicherung' aus der Abbildung E.10 ist über folgende Wörter beschreibbar:

⟨{**sichern**}, T[0] {Z},⟨release({}, 'true'), conclude({Z': 0}, 0 ≤ Z' ≤ 2D)⟩⟩,
if 'Manuell weiter' **then**
⟨{**vor_entsichern**}, T[1], {Z},⟨release({Z: 4*(T[1]-T[0])}, 1D ≤ Z ≤ 2D),
conclude({}, 1D≤ Z')⟩⟩)
else if 'Automatisch weiter' **then**
(
⟨{**entsichern**}, T[1], {Z},⟨release({Z: 4*(T[1]-T[0]}, 1D ≤ Z ≤ 2D+ϵ),
conclude({}, 0 ≤ Z' ≤ 4D)⟩⟩,
⟨{**warten_ende**}, T[2], {Z},⟨release({Z: 4*(T[2]-T[1])}, 2D ≤ Z ≤ 3D),
conclude({}, 1D ≤ Z')⟩⟩
)

Das Verhalten des Automaten 'Intervall' der Abbildung E.8 ist ohne detaillierte Betrachtung der komplexen Lokation 'Warten' über folgende Wörter beschreibbar:

⟨{**initi**}, T[0,0], {Y, Tvor, Tj},⟨release({}, 'true'),
conclude({Y': 0, Tvor': Tj'}, 0 ≤ Y' ∧ Tvor' ≤ Tj' ≤ Tvor'+1)⟩⟩,
⟨{**fertig**}, T[1,0], {Y, Tvor, Tj},
⟨release({Y: 'Y, Tj: 'Tj+(T[1,0]-T[0,0])}, 0 ≤ Y ∧ Tj Tvor+1),
conclude({}, 0 ≤ Y' ≤ 4D ∧ Tvor' ≤ Tj' ≤ Tvor'+1)⟩⟩,
for i: 1 **to** n **do**
(
(⟨{**anfangen**}, T[2,i], {Y, Tvor, Tj},⟨release({Y: 'Y+4*(T[2,i]-T[1,i-1]), Tj: 'Tj},
2D ≤ Y ≤ 3D ∧ Tvor ≤ Tj ≤ Tvor+1),
conclude({Y': 0, Tvor': Tj'}, 0 ≤ Y' ∧ Tvor' ≤ Tj' ≤ Tvor'+1)⟩⟩,
⟨{**fertig**}, T[1,i], {Y, Tvor, Tj},⟨release({Y: 'Y, Tj: 'Tj+(T[1,i]-T[2,i])},
0 ≤ Y ∧ Tj Tvor+1), conclude({}, 0 ≤ Y' ≤ 4D ∧ Tvor' ≤ Tj' ≤ Tvor'+1)⟩⟩,
)
if 'Manuell erzeugter Vorgang' **then**
⟨{**weiter**}, T[2,0], {Y, Tvor, Tj},⟨release({Y: 'Y+4*(T[2,0]-T[1,n]),
Tj: 'Tj+(T[2,0]-T[1,n]}, 0 ≤ Y ≤ 3D ∧ Tvor ≤ Tj ≤ Tvor+1),
conclude({}, 1D ≤ Y')⟩⟩

n kann 0 sein
n -> unendlich
Zyklus 0 bis mehrmals, 'weiter' optional null oder einmal
Wird das Verhalten der komplexen Lokation 'Warten' wie in der Abbildung E.11 detailliert beschrieben, so ergeben sich folgende Wörter:

⟨{**initi**}, T[0,0], {Y, Tvor, Tj},⟨release({}, 'true'),
conclude({Y': 0, Tvor': Tj'}, 0 ≤ Y' ∧ Tvor' ≤ Tj' ≤ Tvor'+1)⟩⟩,
⟨{**fertig, sichern**}, T[1,0], {Y, Tvor, Tj},⟨release({Y: 'Y, Tj: 'Tj+(T[1,0]-T[0,0])},
0 ≤ Y ∧ Tj Tvor+1), conclude({}, 0 ≤ Y' ≤ 2D ∧ Tvor' ≤ Tj' ≤ Tvor'+1)⟩⟩,
for i: 1 **to** n **do**
(
(⟨{**entsichern**}, T[2,i], {Y, Tvor, Tj},⟨release({Y: 'Y+4*(T[2,i]-T[1,i-1]), Tj: 'Tj},
1D ≤ Y ≤ 2D+ϵ ∧ Tvor ≤ Tj ≤ Tvor+1),
conclude({}, 0 ≤ Y' ≤ 4D ∧ Tvor' ≤ Tj' ≤ Tvor'+1)⟩⟩,
⟨{**warten_ende**}, T[3,i], {Y, Tvor, Tj},⟨release({Y: 'Y+4*(T[3,i]-T[2,i]), Tj: 'Tj},
2D ≤ Y ≤ 3D ∧ Tvor ≤ Tj ≤ Tvor+1),
conclude({Y': 0, Tvor': Tj}, 0 ≤ Y' ∧ Tvor' ≤ Tj' ≤ Tvor'+1)⟩⟩,
⟨{**fertig, sichern**}, T[1,i], {Y, Tvor, Tj},⟨release({Y: 'Y, Tj: 'Tj+(T[1,i]-T[3,i]},
0 ≤ Y ∧ Tj Tvor+1), conclude({}, 0 ≤ Y' ≤ 2D ∧ Tvor' ≤ Tj' ≤ Tvor'+1)⟩⟩,
)
if 'Manuell erzeugter Vorgang' **then**
⟨{**vor_entsichern**}, T[2,0], {Y, Tvor, Tj},⟨release({Y: 'Y+4*(T[2,0]-T[1,n]),
Tj: 'Tj+(T[2,0]-T[1,n])}, 0 ≤ Y ≤ 3D ∧ Tvor ≤ Tj ≤ Tvor+1),
conclude({}, 1D ≤ Y')⟩⟩

n kann aber auch 0 sein
n -> unendlich
ϵist der kleinste Wert, der über der angegebenen Intervallgrenze liegt Das Verhalten des Automaten 'Heizen_und_Biegen' ist auf 3 Verfeinerungsebenen beschreibbar. Folgende Wörter symbolisieren das Verhalten ohne Verfeinerung der komplexen Lokation 'Vorheizen' des Automaten in der Abbildung E.7:

⟨{**inithb**}, T[0,0], {X, Tj},⟨release({}, 'true'),
conclude({X': 0, Tj': 15}, $0 \leq X' \wedge Tj' \quad 15$)⟩⟩,
for i: 1 **to** n **do**
(
⟨{**vorheiz**}, T[1,i], {X, Tj},⟨release({X: 'X+(T[1,i]-T[0,i-1]), Tj: 'Tj},
$4 < X < 6 \wedge Tj \quad 15$), conclude({}, $0 \leq X' \wedge 15 \leq Tj' \leq T+4$)⟩⟩,
if 'Automatisches Auslösen' **then**
⟨{**vorheiz_fertig**}, T[2,i], {X, Tj},⟨release({X: 'X+(T[2,i]-T[1,i]),
Tj: 'Tj+(T[2,i]-T[1,i])}, $0 \leq X \wedge T+2 \leq Tj \wedge 15 \leq Tj \wedge Tj \leq T+4+\epsilon$),
conclude({X': 0}, $0 \leq X' \wedge 15 \leq Tj' \leq T+4$)⟩⟩,
if 'Manuelles Auslösen' **then**
⟨{**begin_biegen**}, T[2,i], {X, Tj},⟨release({X: 'X+(T[2,i]-T[1,i]),
Tj: 'Tj+(T[2,i]-T[1,i])}, $0 \leq X \wedge T \leq Tj \wedge 15 \leq Tj \wedge Tj \leq T+2$),
conclude({X': 0}, $0 \leq X' \wedge 15 \leq Tj' \leq T+4$)⟩⟩,
⟨{**biegen_fertig**}, T[3,i], {X, Tj},⟨release({X: 'X+(T[3,i]-T[2,i]), Tj: 'Tj},
$4 \leq X \wedge 15 \leq Tj \leq T+4$), conclude({X': 0}, $0 \leq X' \wedge 15 \leq Tj' \leq T+4$)⟩⟩,
⟨{**neu_einlegen**}, T[0,i], {X, Tj},⟨release({X: T[0,i]-T[3,i], Tj: 'Tj},
$2 \leq X \wedge 15 \leq Tj \leq T+4$), conclude({X': 0,Tj': 15}, $0 \leq X' \wedge Tj' \quad 15$)⟩⟩
)

Ist die Lokation 'Vorheizen' durch den Automatentyp 'Intervall' wie in der Abbildung E.9 verfeinert, so ergeben sich folgende Wörter für das Verhalten:

⟨{**inithb**}, T[0,0,0], {X, Tj},⟨release({}, 'true'),
conclude({X': 0, Tj': 15}, $0 \leq X' \wedge Tj' \quad 15$)⟩⟩,
for i: 1 **to** n **do**
(
⟨{**vorheiz, initi**}, T[1,i,0], {X, Y, Tj, Tvor},
⟨release({X: 'X+(T[1,i,0]-T[0,i-1,0]), Tj: 'Tj}, $4 < X < 6 \wedge Tj \quad 15$),
conclude({Tvor': Tj, Y': 0}, $0 \leq X' \wedge 15 \leq Tj' \leq T+4 \wedge$
$Tvor' \leq Tj' \leq Tvor'+1$)⟩⟩,
if 'Automatisches Auslösen' **then**
(
for j: 1 **to** m **do**
(
⟨{**fertig**}, T[2,i,j], {X, Y, Tj},⟨release({X: 'X+(T[2,i,j]-T[1,i,j-1]), Y: 'Y,

Tj: 'Tj+(T[2,i,j]-T[1,i,j-1])}, $0 \leq X \wedge 0 \leq Y \wedge 15 \leq Tj \leq T+4+\epsilon \wedge$
Tj Tvor+1), conclude({}, $0 \leq X' \wedge 0 \leq Y' \leq 4D \wedge 15 \leq Tj' \leq T+4 \wedge$
Tvor' $\leq Tj' \leq$ Tvor'+1)⟩⟩
⟨{**anfangen**}, T[1,i,j], {X, Y, Tj},⟨release({X: 'X+(T[1,i,j]-T[2,i,j]),
Y: 'Y+4(T[1,i,j]-T[2,i,j]), Tj: 'Tj+(T[1,i,j]-T[2,i,j])},
$0 \leq X \wedge 2D \leq Y \leq 3D \wedge 15 \leq Tj \leq T+4+\epsilon \wedge Tvor \leq Tj \leq Tvor+1+\epsilon$),
conclude({Tvor': Tj, Y': 0}, $0 \leq X' \wedge 0 \leq Y' \wedge 15 \leq Tj' \leq T+4 \wedge$
$0 \leq X' \wedge 0 \leq Y' \wedge 15 \leq Tj' \leq T+4 \wedge$Tvor' $\leq Tj' \leq$ Tvor'+1)⟩⟩
)
⟨{**fertig**}, T[2,i,0], {X, Y, Tj},⟨release({X: 'X+(T[2,i,0]-T[1,i,m]), Y: 'Y,
Tj: 'Tj+(T[2,i,0]-T[1,i,m])}, $0 \leq X \wedge 0 \leq Y \wedge 15 \leq Tj \leq T+4+\epsilon \wedge$
T+2 $\leq Tj \wedge$ Tj Tvor+1), conclude({X': 0}, $0 \leq X' \wedge 15 \leq Tj' \leq T+4$)⟩⟩)
)
if 'Manuelles Auslösen' **then**
(
⟨{**fertig**}, T[1,i,1], {X, Y, Tj},⟨release({X: 'X+(T[1,i,1]-T[1,i,0]), Y: 'Y,
Tj: 'Tj+(T[1,i,1]-T[1,i,0])}, $0 \leq X \wedge 0 \leq Y \wedge 15 \leq Tj \leq T+4+\epsilon \wedge$
Tj Tvor+1), conclude({}, $0 \leq X' \wedge 0 \leq Y' \leq 4D \wedge 15 \leq Tj' \leq T+4 \wedge$
Tvor' $\leq Tj' \leq$ Tvor'+1)⟩⟩
for k: 2 **to** o **do**
(
⟨{**anfangen**}, T[2,i,k], {X, Y, Tj},⟨release({X: 'X+(T[2,i,k]-T[1,i,k-1]),
Y: 'Y+4(T[2,i,k]-T[1,i,k-1]), Tj: 'Tj+(T[2,i,k]-T[1,i,k-1])},
$0 \leq X \wedge 2D \leq Y \leq 3D \wedge 15 \leq Tj \leq T+4+\epsilon \wedge Tvor \leq Tj \leq Tvor+1+\epsilon$),
conclude({Tvor': Tj, Y': 0}, $0 \leq X' \wedge 0 \leq Y' \wedge 15 \leq Tj' \leq T+4 \wedge$
Tvor' $\leq Tj' \leq$ Tvor'+1)⟩⟩
⟨{**fertig**}, T[1,i,k], {X, Y, Tj},⟨release({X: 'X+(T[1,i,k]-T[2,i,k]), Y: 'Y,
Tj: 'Tj+(T[1,i,k]-T[2,i,k])}, $0 \leq X \wedge 0 \leq Y \wedge 15 \leq Tj \leq T+4+\epsilon \wedge$
Tj Tvor+1), conclude({}, $0 \leq X' \wedge 0 \leq Y' \leq 4D \wedge 15 \leq Tj' \leq T+4 \wedge$
Tvor' $\leq Tj' \leq$ Tvor'+1)⟩⟩
)
⟨{**weiter**}, T[2,i,0], {X, Y, Tj},⟨release({X: 'X+(T[2,i,0]-T[1,i,o+1]),
Y: 'Y+4(T[2,i,0]-T[1,i,o+1]), Tj: 'Tj+(T[2,i,0]-T[1,i,o+1])}, $0 \leq X \wedge$
1D $\leq Y \leq 3D \wedge 15 \leq Tj \wedge T \leq Tj \leq T+2 \wedge Tvor \leq Tj \leq Tvor+1+\epsilon$),
conclude({X': 0}, $0 \leq X' \wedge 15 \leq Tj' \leq T+4$)⟩⟩
)
⟨{**biegen_fertig**}, T[3,i,0], {X, Tj},⟨release({X: 'X+(T[3,i,0]-T[2,i,0]), Tj: 'Tj},
$4 \leq X \wedge 15 \leq Tj \leq T+4$), conclude({X': 0}, $0 \leq X' \wedge 15 \leq Tj' \leq T+4$)⟩⟩,
⟨{**neu_einlegen**}, T[0,i,0], {X, Tj},⟨release({X: T[0,i,0]-T[3,i,0], Tj: 'Tj},
$2 \leq X \wedge 15 \leq Tj \leq T+4$), conclude({X': 0,Tj': 15}, $0 \leq X' \wedge$ Tj' 15)⟩⟩
)

n, m, o können 0 sein
Im Fall einer weiteren Verfeinerung der Lokation 'Warten' des Automatentyps 'Intervall' durch den Automatentyp 'Sicherung' wie in der Abbildung E.12 basiert das Verhalten des gesamten Automaten 'Heizen_und_Biegen' auf folgenden Wörtern:

⟨{**inithb**}, T[0,0,0], {X, Tj},⟨release({}, 'true'),
conclude({X': 0, Tj': 15}, 0 ≤ X' ∧ Tj' 15)⟩⟩,
for i: 1 **to** n **do**
(
⟨{**vorheiz, initi**}, T[1,i,0], {X, Y, Tj, Tvor},⟨release({X: 'X+(T[1,i,0]-T[0,i-1,0]),
Tj: 'Tj}, 4 < X < 6 ∧ Tj 15), conclude({Tvor': Tj, Y': 0}, 0 ≤ X' ∧
15 ≤ Tj' ≤ T+4 ∧ Tvor'≤ Tj' ≤ Tvor'+1)⟩⟩,
if 'Automatisches Auslösen' **then**
(
for j: 1 **to** m **do**
(
⟨{**fertig,sichern**}, T[3,i,j], {X, Y, Tj, Tvor},
⟨release({X: 'X+(T[3,i,j]-T[1,i,j-1]), Y: 'Y, Tj: 'Tj+(T[2,i,j]-T[1,i,j-1])},
0 ≤ X ∧ 0 ≤ Y ∧ 15 ≤ Tj ≤ T+4+ϵ ∧ Tj Tvor+1), conclude({}, 0 ≤ X' ∧
0 ≤ Y' ≤ 2D ∧ 15 ≤ Tj' ≤ T+4 ∧ Tvor' ≤ Tj' ≤ Tvor'+1)⟩⟩
⟨{**entsichern**}, T[2,i,j], {X, Y, Tj, Tvor},⟨release({X: 'X+(T[2,i,j]-T[3,i,j]),
Y: 'Y+4(T[2,i,j]-T[3,i,j]), Tj: 'Tj}, 0 ≤ X ∧ 1D ≤ Y ≤ 2D ∧
15 ≤ Tj ≤ T+4 ∧ Tvor ≤ Tj ≤ Tvor+1), conclude({}, 0 ≤ X' ∧
0 ≤ Y' ≤ 4D ∧ 15 ≤ Tj' ≤ T+4 ∧ Tvor' ≤ Tj' ≤ Tvor'+1)⟩⟩
⟨{**warten_ende**}, T[1,i,j], {X, Y, Tj, Tvor},⟨release({X: 'X+(T[1,i,j]-T[2,i,j]),
Y: 'Y+4(T[1,i,j]-T[2,i,j]), Tj: 'Tj}, 0 ≤ X ∧ 2D ≤ Y ≤ 3D ∧
15 ≤ Tj ≤ T+4 ∧Tvor ≤ Tj ≤ Tvor+1), conclude({Tvor': Tj, Y': 0},
0 ≤ X' ∧ 0 ≤ Y' ∧ 15 ≤ Tj' ≤ T+4 ∧ Tvor' ≤ Tj' ≤ Tvor'+1)⟩⟩
)
⟨{**fertig,sichern**}, T[2,i,0], {X, Y, Tj, Tvor},⟨release({X: 'X+(T[2,i,0]-T[1,i,m]),
Y: 'Y, Tj: 'Tj+(T[2,i,0]-T[1,i,m])}, 0 ≤ X ∧ 0 ≤ Y ∧ 15 ≤ Tj ≤ T+4+ϵ ∧
T+2 ≤ Tj ∧ Tj Tvor+1), conclude({X': 0}, 0 ≤ X' ∧ 15 ≤ Tj' ≤ T+4)⟩⟩)
)
if 'Manuelles Auslösen' **then**
(
⟨{**fertig,sichern**}, T[1,i,1], {X, Y, Tj, Tvor},⟨release({X: 'X+(T[1,i,1]-T[1,i,0]),
Y: 'Y, Tj: 'Tj+(T[1,i,1]-T[1,i,0])}, 0 ≤ X ∧ 0 ≤ Y ∧ 15 ≤ Tj ≤ T+4+ϵ ∧
Tj Tvor+1), conclude({}, 0 ≤ X' ∧ 0 ≤ Y' ≤ 2D ∧ 15 ≤ Tj' ≤ T+4 ∧
Tvor' ≤ Tj' ≤ Tvor'+1)⟩⟩
for k: 2 **to** o **do**
(
⟨{**entsichern**}, T[3,i,k], {X, Y, Tj, Tvor},⟨release({X: 'X+(T[3,i,k]-T[1,i,k-1]),
Y: 'Y+4(T[3,i,k]-T[1,i,k-1]), Tj: 'Tj}, 0 ≤ X ∧ 1D ≤ Y ≤ 2D+ϵ ∧

$15 \leq Tj \leq T+4 \wedge Tvor \leq Tj \leq Tvor+1)$, conclude({}, $0 \leq X' \wedge$
$0 \leq Y' \leq 4D \wedge 15 \leq Tj' \leq T+4 \wedge Tvor' \leq Tj' \leq Tvor'+1)\rangle\rangle$
$\langle${**warten_ende**}, T[2,i,k], {X, Y, Tj, Tvor},$\langle$release({X: 'X+(T[2,i,k]-T[3,i,k]),
Y: 'Y+4(T[2,i,k]-T[3,i,k]), Tj: 'Tj}, $0 \leq X \wedge 2D \leq Y \leq 3D \wedge$
$15 \leq Tj \leq T+4 \wedge Tvor \leq Tj \leq Tvor+1)$, conclude({Tvor': Tj, Y': 0},
$0 \leq X' \wedge 0 \leq Y' \wedge 15 \leq Tj' \leq T+4 \wedge Tvor' \leq Tj' \leq Tvor'+1)\rangle\rangle$
$\langle${**fertig,sichern**}, T[1,i,k], {X, Y, Tj, Tvor},$\langle$release({X: 'X+(T[1,i,k]-T[2,i,k]),
Y: 'Y, Tj: 'Tj+(T[1,i,k]-T[2,i,k])}, $0 \leq X \wedge 0 \leq Y \wedge 15 \leq Tj \leq T+4+\epsilon \wedge$
Tj Tvor+1), conclude({}, $0 \leq X' \wedge 0 \leq Y' \leq 2D \wedge 15 \leq Tj' \leq T+4 \wedge$
$Tvor' \leq Tj' \leq Tvor'+1)\rangle\rangle$
)
$\langle${**vor_entsichern**}, T[2,i,0], {X, Y, Tj, Tvor},
$\langle$release({X: 'X+(T[2,i,0]-T[1,i,o+1]), Y: 'Y+4(T[2,i,0]-T[1,i,o+1]), Tj: 'Tj},
$0 \leq X \wedge 1D \leq Y \leq 2D \wedge 15 \leq Tj \wedge T \leq Tj \leq T+2 \wedge Tvor \leq Tj \leq Tvor+1)$,
conclude({X': 0}, $0 \leq X' \wedge 15 \leq Tj' \leq T+4)\rangle\rangle$
)
$\langle${**biegen_fertig**}, T[3,i,0], {X, Tj}, $\langle$release({X: 'X+(T[3,i,0]-T[2,i,0]), Tj: 'Tj},
$4 \leq X \wedge 15 \leq Tj \leq T+4)$, conclude({X': 0}, $0 \leq X' \wedge 15 \leq Tj' \leq T+4)\rangle\rangle$,
$\langle${**neu_einlegen**}, T[0,i,0], {X, Tj},$\langle$release({X: T[0,i,0]-T[3,i,0], Tj: 'Tj},
$2 \leq X \wedge 15 \leq Tj \leq T+4)$, conclude({X': 0,Tj': 15}, $0 \leq X' \wedge Tj'$ 15)$\rangle\rangle$
)

E.3.4 Hybrides System

Das hybride System besteht aus zwei Phasen, einer Phase zur Reinigung des Bambusstabes und einer weiteren Phase zum Vorheizen und Biegen des Bambusstabes. Die Phasen hängen in der Ausführung von festgelegten Parametern für den Durchmesser eines Bambusstabes und die zu errreichende Temperatur ab. Beide Phasen synchronisieren sich zur Durchführung eines kontinuierlichen Ablaufes über Signale. Die Signale werden untereinander und mit der Umgebung ausgetauscht.

Notation in MODEL-HS und VYSMO

In MODEL-HS wird das gesamte System mit folgendem Programmtext beschrieben:

system Bambus_Bearbeitung (**parameter** T, D;
signal in erst_ein, ein_u_bieg, aktiv_r, manuell
out gereinigt, Phase1.einlegen, aktiv_Int,
heizen, anfang, sichern, fertig,
entsichern, heiz_fertig, bieg_fertig);

```
import from Library
    automaton_import
        Reinigung(invariant ReinInv;
                  activity ReinAct;
                  parameter DM;
                  signal synchron in   eingelegt
                                out    sauber
                         refine   in    initr
                                out    neu_einlegen);
        Heizen_und_Biegen(invariant HeizInv;
                          activity HeizAct;
                          parameter T,D;
                          signal synchron in   inithb, begin_biegen
                                       out    vorheiz, initi, anfangen,
                                              sichern, fertig, entsichern,
                                              vorheiz_fertig, biegen_fertig
                                 refine   in   neu_einlegen);

declaration
    automata
        Phase_1: Reinigung;
        Phase_2: Heizen_und_Biegen;
    clock S;
    signal heiz_fertig, einlegen;

synchronisation
    initialisation
        T ≤ 15;
        S :  0;
    connection
        Phase_1(0 ≤ S, {S:  S+1}, D, (erst_ein(N  0,N:  1) ∨ (heiz_fertig(S:  0) ∧ ein_u_bieg)),
                gereinigt, aktiv_r, einlegen(S<15)) |
        Phase_2(0 ≤ S, {S:  S+1}, T, D, Phase_1.einlegen(S:  0), manuell, heizen,
                aktiv_int, anfang, sichern, fertig, entsichern, heiz_fertig(S<8),
                bieg_fertig, Phase_1.einlegen(S:  0))
endsystem Bambus_Bearbeitung.
```

In VYSMO ist das hybride System wie in Abbildung E.13 dargestellt.
In der Parameterliste des Systems werden mit der Temperatur T und dem Durchmesser D sowie den Signalen, welche vom System aus der Umgebung empfangen und an die Umgebung gesendet werden, alle Parameter aufgeführt, die die Schnittstelle für eine weitere Modularisierung in übergeordnenten Systemen bilden. Im System 'Bambus_Bearbeitung' werden die zwei Automatentypen 'Reinigung' und 'Heizen_und_Biegen' importiert, wel-

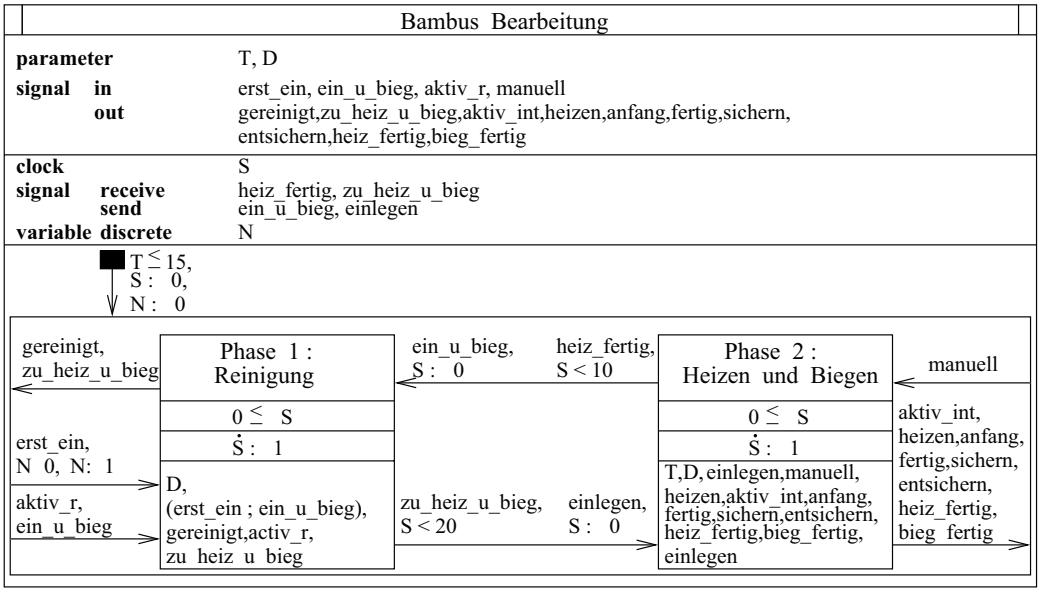


Abbildung E.13: Synchronisierend hybrider Automat des Systems 'Bambus_Bearbeitung'

che durch die Bearbeitungsphasen 'Phase_1' und 'Phase_2' instanziiert sind. In dem System ist eine Uhr 'S' erklärt, welche im Rhythmus der Systemzeit tickt. Die Signale 'heiz_fertig' und 'einlegen' sowie deren Umbenennungsvarianten 'ein_u_bieg' und 'zu_heiz_u_bieg' aus Abbildung E.13 dienen der Synchronisation zwischen den Automaten innerhalb des Systems und werden deshalb als lokal deklarierte Signale aufgeführt.
Der eigentliche Synchronisationsvorgang zwischen den Automaten 'Phase_1', 'Phase_2' und der Umgebung wird mit der für alle Unterautomaten gültigen Anfangsbedingung 'T $\leq$ 15' and Zuweisung 'S: 0' an dem Anfangsübergang eingeleitet. Der Austausch der Signale zur Synchronisation ist mit Zeitbedingungen und Zuweisungen neuer Werte der Systemuhr 'S' verbunden. Die Synchronisationsverbindung mit dem Signal 'heiz_fertig' ist dabei zum Beispiel mit der Zeitbedingung 'S > 8' und der Zuweisung 'S: 0' verbunden und vom Automaten 'Phase_1' kann das Signal 'einlegen' nur ausgesendet werden, solange die Uhr 'S' einen Wert kleiner als 15 beinhaltet.
In diesem System ist bezüglich der Signale 'heiz_fertig' und 'ein_u_bieg' eine Besonderheit gegeben. Die Reinigungsphase synchronisiert sich sowohl mit der Umgebung als auch mit der Phase für das Heizen und Biegen. Aus der Umgebung wird ein neuer Bambusstab geliefert, der gereinigt werden soll. Dabei werden zwei Fälle unterschieden. Nach der Inbetriebnahme des Systems wird das Einlegen des ersten Bambusstabes zur Reinigung mit dem Signal 'erst_ein' abgschlossen. Danach soll das Einlegen eines neuen Stabes für die Reinigung mit dem Ende des Aufheizens des Vorgängerstabes aus dem Prozess 'Heizen_und_Biegen' zusammenfallen, wodurch das Signal 'ein_u_bieg' im synchronisierend hybriden Automaten der VYSMO Beschreibung zweimal auftritt.

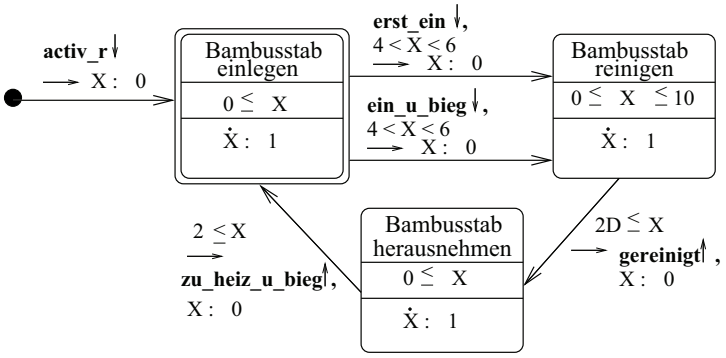


Abbildung E.14: Verhaltensbeschreibung des Automaten 'Reinigung'

Spezifizierte Wörter

Die Beschreibung möglicher Wörter des gesamten Systems lässt sich gut an der Konstruktion des Produktautomaten für 'Bambus_Bearbeitung' nachvollziehen. Da dieser Automat für eine übersichtliche Darstellung zu komplex ist und auch während der symbolischen Simulation nur der für eine spezifizierte Eigenschaft relevante Teilautomat durchlaufen wird, beschränkt sich die Darstellung an dieser Stelle auf einen kleinen Auszug des zu erzeugenden Produktes zur Synchronisation.

In den Abbildungen E.14 und E.15 sind zu besseren Bezugnahme die Verhaltensbeschrei-

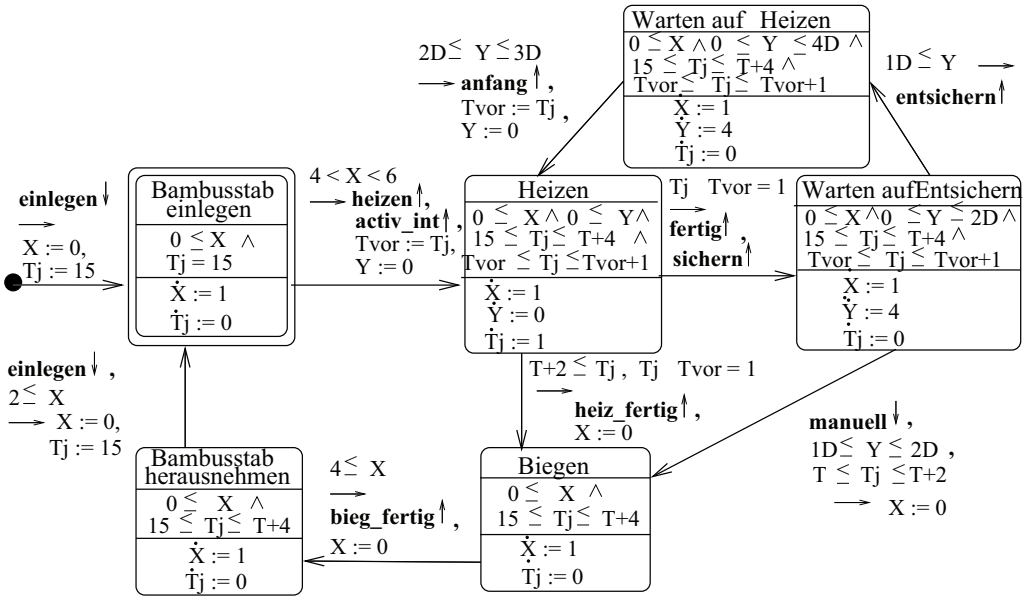


Abbildung E.15: Verhaltensbeschreibung des Automaten 'Heizen_und_Biegen'

bungen der Automaten 'Reinigung' und 'Heizen_und_Biegen' der Abbildungen E.6 und

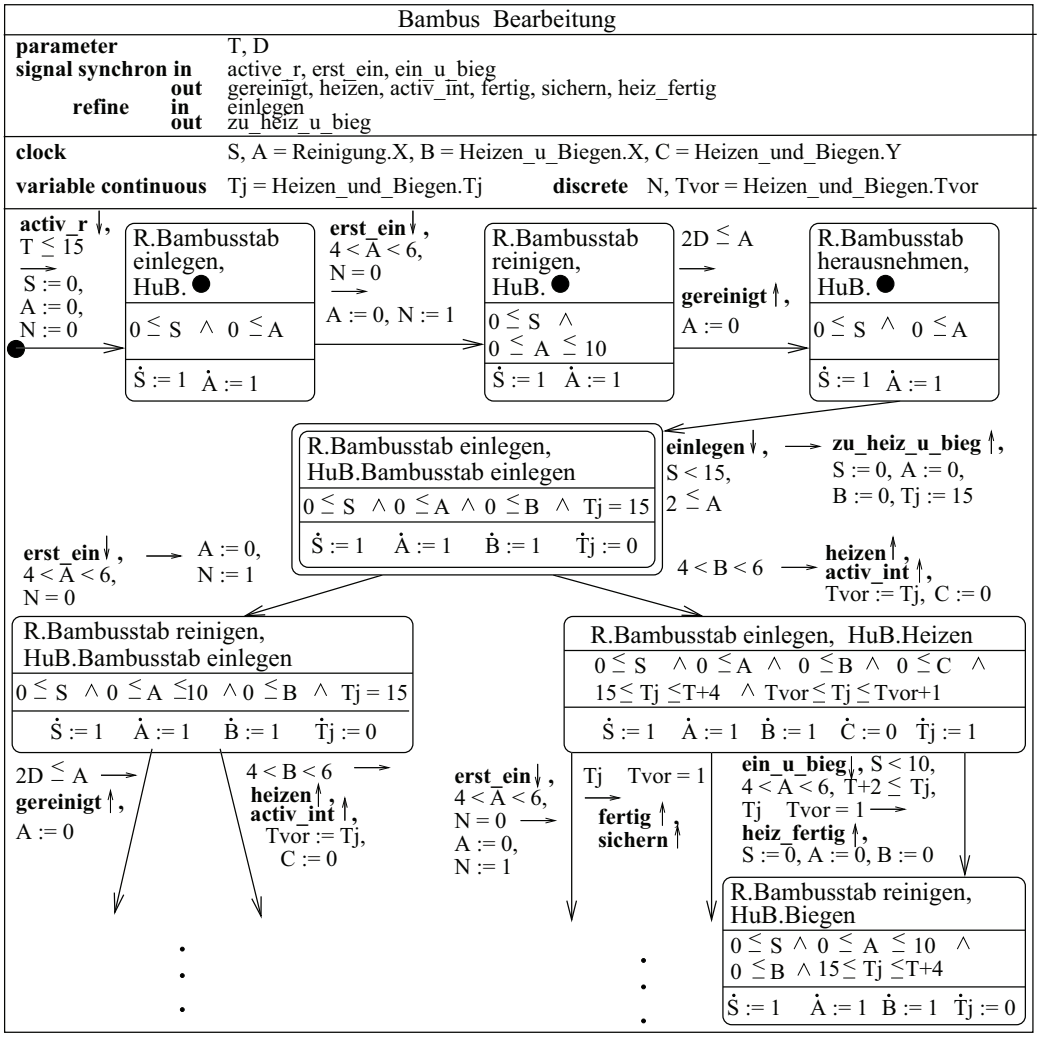


Abbildung E.16: Auszug aus dem Synchronisationsprodukt

E.12 mit den aktuell übergebenen Signalen aus dem Automaten ’Bambus_Bearbeitung’ der Abbildung E.13 angegeben.

Produktautomat

In der Abbildung E.16 ist ein kleiner Ausschnitt des Produktautomaten gezeigt, der durch die Synchronisation des Automaten ’Reinigung’ mit dem Automaten ’Heizen_und_Biegen’ entsteht. Zur besseren Handhabung wurden für die Uhren und Variablen der Automaten die kürzeren Bezeichner ’A’, ’B’, ’C’, ’Tj’ und ’Tvor’ verwendet. wesentlich, wenn Invarianten von Lokationen keinen Zeitverzug erzwingen, können sequentiell aufgeführte Signale massiv parallel erfolgen, also auch in Wörtern auftreten.

Vollständiges Synchronisationsprodukt

In Abbildung E.17 ist ein vollständiges Synchronisationsprodukt mit all denjenigen Lokationen und Transitionen ohne Invarianten, Aktivitäten, Transitionsbedingungen und Aktionen erstellt worden, die erreicht werden, da der Übergang mit dem Signal 'erst_ein' aufgrund der Anfangsbedingung 'N 0' der Abbildung E.13 kein zweites Mal ausgeführt werden kann.

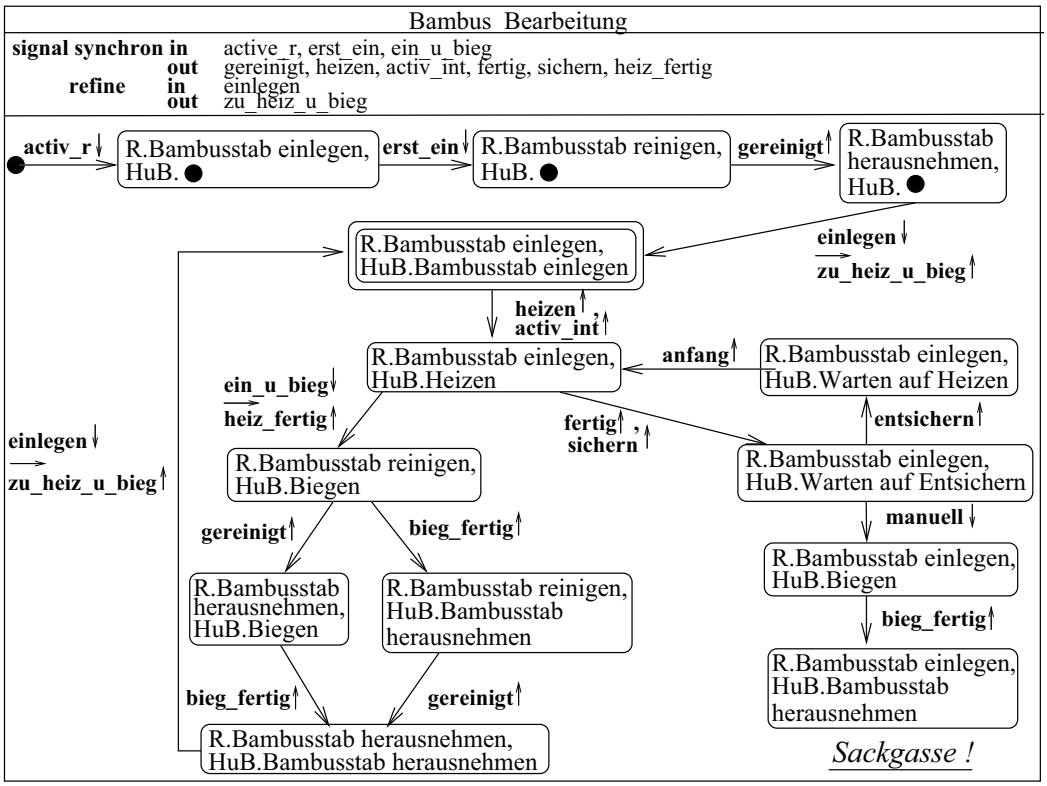


Abbildung E.17: Synchronisationsprodukt ohne Invarianten, Aktivitäten, Transitionsbedingungen und Aktionen

Allein aus der Synchronisation durch die Signale an den einzelnen Transitionen ergibt sich ein Pfad, der nicht zur Akzeptanz eines Wortes führt. Sobald in der Lokation 'Warten_auf_Entsichern' des Automaten 'Heizen_und_Biegen' die Entsicherung manuell vorgenommen wird, kann nach einem erfolgreichen Biegen des Bambusstabes weder ein Herausnehmen des Stabes noch ein Einlegen eines neuen Stabes zur Reinigung erfolgen, da die Synchronisation der Signale 'ein_u_bieg' mit 'heiz_fertig' nicht stattfinden konnte.

Anhang F

Symbolische Simulation

F.1 Regeln und Semantik

Die Prädikate 'release' und 'conclude' werden durch folgende Ziele bestimmt:

release(*Symbole, T, Tvor, VAR, VARvor, Aktiv_Lok, Nachf_Lok, Bedingung*⟩**,**
*Aktivitäten, UnbedingtTrans, BedingtTrans***)**
:-
activities(*T, Tvor, VAR, VARvor, Aktiv_Lok, Aktivitäten***),**
uncond_trans(*Symbole, VAR, Aktiv_Lok, Nachf_Lok, UnbedingtTrans***),**
cond_trans(*Symbole, VAR, BedingtTrans***).**

conclude(*Symbole, VAR, VARnach, Aktiv_Lok, Aktionen, EintrittInv***)**
:-
actions(*Symbole, VAR, VARnach, Aktionen***),**
entrance_cond(*VARnach, Nachf_Lok, EintrittInv***).**

Im Klauselrumpf von 'release' werden über die Ziele 'activities', 'uncond_trans' und 'cond_trans' die Aktivitäten bezüglich der gerade aktiven Lokation, die unbedingten Transitionsbedingungen und die bedingten Transitionsbedingungen ermittelt. Werte von Variablen, für die keine Aktivitäten angegeben sind, werden aus der Menge 'VARvor' in 'VAR' kopiert.
Im Klauselrumpf von 'conclude' werden über die Ziele 'actions' und 'entrance_cond' alle Aktionen und Bedingungen zum Eintreten in die nachfolgende Lokation ermittelt. Werte von Variablen, die nicht durch Aktionen beeinflusst sind, werden von der Menge 'VAR' in die Menge 'VARnach' übernommen.

F.2 Abhängigkeiten und Bindungen

Constraints werden während der symbolischen Simulation gesammelt und zu einem Constraintsystem zusammengefasst, welches durch einen gegebenen Constraintlöser bis zur Lösung vereinfacht wird. Dabei beeinflussen die Bedingungen eines Aufrufes sowohl die Variablenbelegungen vorhergehender Aufrufe als auch nachfolgender Aufrufe der Klausel von 'symb_sim'. Beziehungen zwischen Variablenbelegungen rückbezüglich vorhergehender Aufrufe ist in der Abbildung F.1 dargestellt.

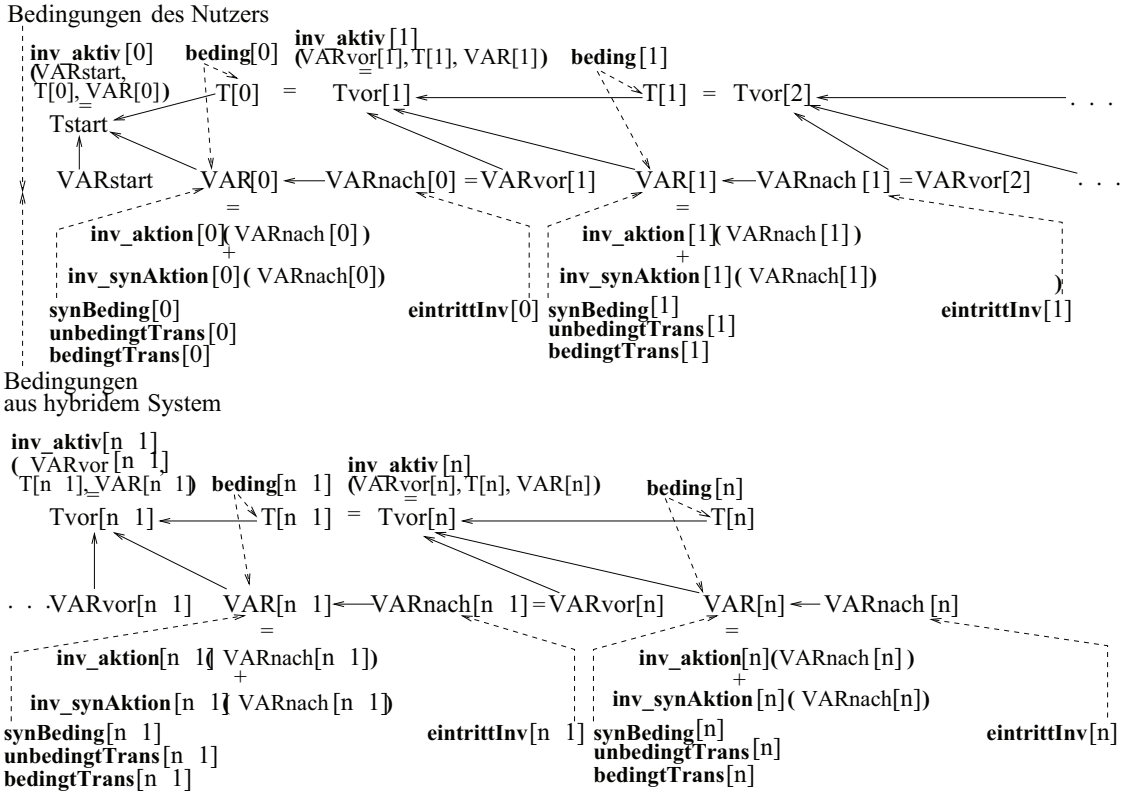


Abbildung F.1: Auswirkungen auf Variablen und Bedingungen rückbezüglich vorhergehender Aufrufe

In der Abbildung F.2 sind die Abhängigkeiten der Zeiten 'T[i]', i 0,..,n, von der Zeit 'Tvor[i]' und den Variablen 'VARvor[i]' bzw. 'VAR[i]' dargestellt. Dabei ergibt sich die Funktion 'time_aktiv' als inverse Funktion zu 'aktivitäten' aus der Abbildung 5.2. Diese Beziehung ist von wesentlicher Bedeutung, da aus den gegebenen Bedingungen, welche im hybriden System für die Variablen bestehen, Rückschlüsse auf Bedingungen für die Zeitpunkte der Akzeptanz der Symbole des Zeitwortes gezogen werden sollen.

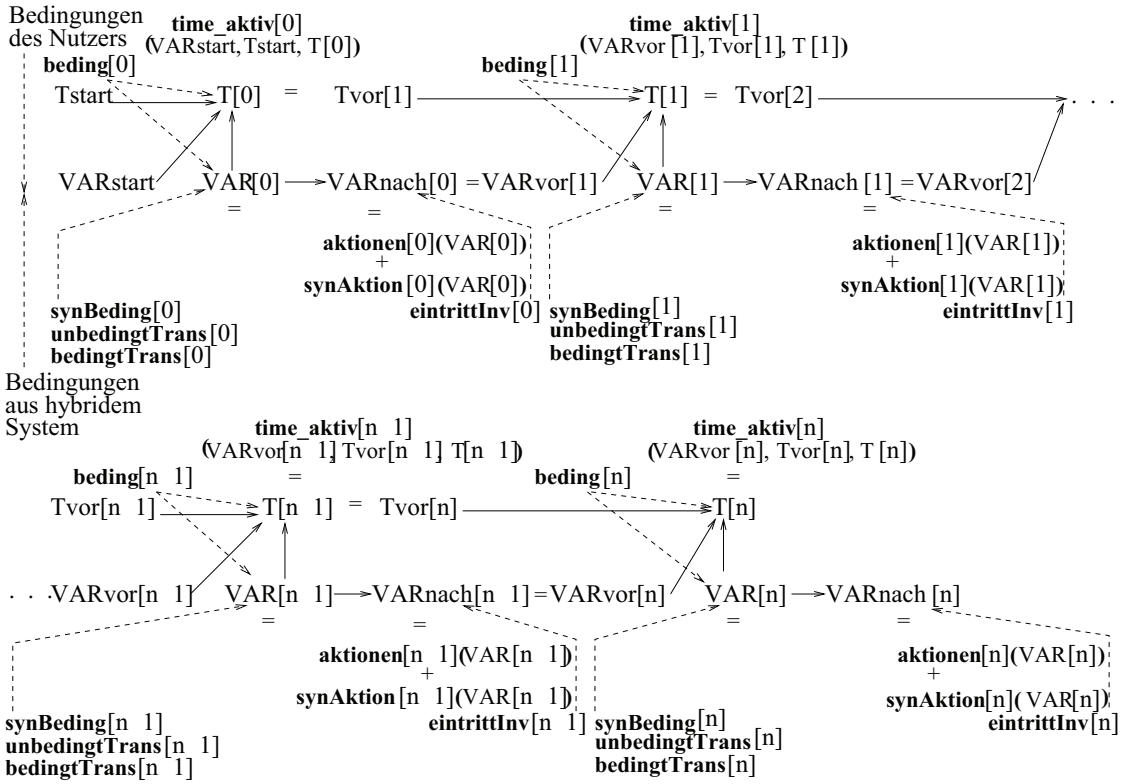


Abbildung F.2: Abhängigkeiten von Variablen und Bindung von Bedingungen

Literaturverzeichnis

[AAB+07] R. Alur, M. Arenas, P. Barceló, K. Etessami, N. Immerman, and L. Libkin. First-Order and Temporal Logics for Nested Words. In *LICS*, pages 151 160. IEEE Computer Society, 2007.

[ÁBKS05] E. Ábrahám, B. Becker, F. Klaedtke, and M. Steffen. Optimizing Bounded Model Checking for Linear Hybrid Systems. In *VMCAI*, volume 3385 of *LNCS*, pages 396 412. Springer, 2005.

[ACD90] R. Alur, C. Courcoubetis, and D.L. Dill. Model-Checking for Real-Time Systems. In *5th Annual Symposium on Logic in Computer Science,* Philadelphia, USA, LICS, pages 414 425. IEEE Society Press, 1990.

[ACD93] R. Alur, C. Courcoubetis, and D. Dill. Model-Checking in Dense Real-Time. *Information and Computation*, 104(1):2 34, May 1993.

[ACH+95] R. Alur, C. Courcoubetis, N. Halbwachs, T.A. Henzinger, X. Nicollin P.-H. Ho, A. Olivero, J. Sifakis, and S. Yovine. The Algorithmic Analysis of Hybrid Systems. In *TCS*, number 138 in TCS, pages 3 34. Elsevier Science, 1995.

[ACHH93] R. Alur, C. Courcoubetis, T.A. Henzinger, and P.-H. Ho. Hybrid Automata: An Approach to the Specification and Verification of Hybrid Systems. In *Hybrid Systems*, number 736 in LNCS, pages 209 229. Springer, 1993.

[ACKS02] G. Audemard, A. Cimatti, A. Kornilowicz, and R. Sebastiani. Bounded Model Checking for Timed Systems. In *FORTE '02*, pages 243 259, London, UK, 2002. Springer-Verlag.

[ACM06] R. Alur, S. Chaudhuri, and P. Madhusudan. Languages of Nested Trees. In Thomas Ball and Robert B. Jones, editors, *CAV*, volume 4144 of *LNCS*, pages 329 342. Springer, 2006.

[AD94] R. Alur and D. L. Dill. A Theory of Timed Automata. *Theoretical Computer Science*, 126(2):183 235, 1994.

[ADE+03] R. Alur, T. Dang, J. M. Esposito, Y. Hur, F. Ivancic, V. Kumar, I. Lee, P. Mishra, G. J. Pappas, and O. Sokolsky. Hierarchical Modeling and Analysis of Embedded Systems. *IEEE*, 91(1):11 28, 2003.

[ADI06] R. Alur, T. Dang, and F. Ivančiacutec. Predicate Abstraction for Reachability Analysis of Hybrid Systems. *ACM Transactions of Embedded Computer System*, 5:152 199, 2006.

[ADM02] E. Asarin, T. Dang, and O. Maler. The d/dt Tool for Verification of Hybrid Systems. In *CAV'02*, volume 2404 of *LNCS*, pages 365 370. Springer, 2002.

[AG04] R. Alur and R. Grosu. Modular Refinement of Hierarchic Reactive Machines. *ACM Transaction Programming Languages and Systems*, 26(2):339 369, 2004.

[AGH+00] R. Alur, R. Grosu, Y. Hur, V. Kumar, and I. Lee. Modular Specification of Hybrid Systems in CHARON. In *Hybrid Systems: Computation and Control, Proceedings of the Third International Conference, (HSCC'00)*, number 1790 in LNCS, pages 6 19. Springer, 2000.

[AGLS01] R. Alur, R. Grosu, I. Lee, and O. Sokolsky. Compositional Refinement of Hierarchical Hybrid Systems. In *Fourth International Workshop on Hybrid Systems: Computation and Control*, number 2034 in LNCS, pages 33 48. Springer, 2001.

[AGLS06] R. Alur, R. Grosu, I. Lee, and O. Sokolsky. Compositional Modeling and Refinement for Hierarchical Hybrid Systems. *Journal of Logic and Algebraic Programming*, 68(1-2):105 128, July 2006.

[AH89] R. Alur and T.A. Henzinger. A Really Temporal Logic. In *30th Annual Symposium on Foundations of Computer Science*, pages 164 169. IEEE Computer Society Press, 1989.

[AH92] R. Alur and T.A. Henzinger. Logics and Models of Real Time: a Survey. In *REX Workshop on Real-Time: Theory in Practice*, volume 600 of *LNCS*, pages 74 106. Springer, 1992.

[AH93] R. Alur and T.A. Henzinger. Real-Time Logics: Complexity and Expressiveness. *Information and Computation*, 104(1):35 77, 1993.

[AHH96] R. Alur, T.A. Henzinger, and P.-H. Ho. Automatic Symbolic Verification of Embedded Systems. In *IEEE Transactions on Software Engineering*, number 22 in IEEE, pages 181 201, 1996.

[AIK+03] R. Alur, F. Ivancic, J. Kim, I. Lee, and O. Sokolsky. Generating Embedded Software from Hierarchical Hybrid Models. *SIGPLAN Notices*, 38(7):171 182, 2003.

[AILS07] L. Aceto, A. Ingólfsdóttir, K.G. Larsen, and J. Srba. *Reactive Systems - Modeling, Specification and Verification*. Cambridge University Press, 2007.

[AKY99] R. Alur, S. Kannan, and M. Yannakakis. Communicating Hierarchical State Machines. In *Automata, Languages, and Programming: Proceedings of the 26th International Conference, (ICALP'99)*, number 1644 in LNCS, pages 169 178. Springer, 1999.

[Alu99] R. Alur. Timed Automata. In *CAV'99*, volume 1633 of *LNCS*, pages 8 22. Springer, 1999.

[AM06] R. Alur and P. Madhusudan. Adding Nesting Structure to Words. In Oscar H. Ibarra and Zhe Dang, editors, *Developments in Language Theory*, volume 4036 of *LNCS*, pages 1 13. Springer, 2006.

[AMP95] E. Asarin, O. Maler, and A. Pnueli. Reachability analysis of dynamical systems having piecewise-constant derivatives. In *TCS*, number 138 in TCS, pages 35 65. Elsevier Science, 1995.

[AMPS08] E. Asarin, V. Mysore, A. Pnueli, and G. Schneider. Low Dimensional Hybrid Systems: Decidable, Undecidable, Don't Know. Journal paper, to be submitted soon to *Information and Computation*, 2008.

[APSY02] E. Asarin, G. J. Pace, G. Schneider, and S. Yovine. SPeeDI - A Verification Tool for Polygonal Hybrid Systems. In *CAV'02*, volume 2404 of *LNCS*, pages 354 358. Springer, 2002.

[AS02] E. Asarin and G. Schneider. Widening the Boundary between Decidable and Undecidable Hybrid Systems. In *CONCUR '02: 13th International Conference on Concurrency Theory*, pages 193 208, London, UK, 2002. Springer-Verlag.

[ASK04] A. Agrawal, G. Simon, and G. Karsai. Semantic Translation of Simulink/Stateflow Models to Hybrid Automata Using Graph Transformations. *Electronic Notes in Theoretical Computer Science*, 109:43 56, 2004.

[AY01] R. Alur and M. Yannakakis. Model Checking of Hierarchical State Machines. *ACM Transactions on Programming Languages and Systems*, 23(3):273 303, 2001.

[Bar92] P. I. Barton. *The Modelling and Simulation of Combined Discrete/Continuous Processes*. PhD thesis, University of London, 1992.

[BBE+04] B. Becker, M. Behle, F. Eisenbrand, M. Fränzle, M. Herbstritt, C. Herde, J. Hoffmann, D. Kröning, B. Nebel, Ilia Polian, and Ralf Wimmer. Bounded Model Checking and Inductive Verification of Hybrid Discrete-continuous Systems. In *ITG/GI/GMM-Workshop,Methoden und Beschreibungssprachen zur Modellierung und Verifikation von Schaltungen und Systemen*, pages 65 75, 2004.

[BBF+99] B. Bérard, M. Bidoit, A. Finkel, F. Laroussinie, A. Petit, L. Petrucci, Ph. Schnoebelen, and P. McKenzie. *Systems and Software Verification - Model-Checking Techniques and Tools*. Springer, Paris, 1999.

[BBHP06] K. Berkenkötter, Stefan Bisanz, Ulrich Hannemann, and Jan Peleska. The HybridUML Profile for UML 2.0. *Int. Journal of Software Tools Technology Transfer*, 8(2):167 176, 2006.

[BBP+02] K. Bender, M. Broy, I. Péter, A. Pretschner, and T. Stauner. Model Based Development of Hybrid Systems: Specification, Simulation, Test Case Generation. In *Modelling, Analysis, and Design of Hybrid Systems*, volume 279 of *LNCIS*, pages 37 52. Springer, July 2002.

[BCC+03] A. Biere, A. Cimatti, E. M. Clarke, O. Strichman, and Y. Zhu. Bounded Model Checking. *Advances in Computers*, 58:118 149, 2003.

[BCCZ99] A. Biere, A. Cimatti, E. M. Clarke, and Y. Zhu. Symbolic Model Checking without BDDs. In *TACAS '99*, volume 1579 of *LNCS*, pages 193 207. Springer, 1999.

[BCD+06] M. Broy, M. L. Crane, J. Dingel, A. Hartman, B. Rumpe, and B. Selic. 2nd UML 2 Semantics Symposium: Formal Semantics for UML. In T. Kühne, editor, *MoDELS 2006 Workshops*, volume 4364 of *LNCS*, pages 318 323. Springer, 2006.

[BCM+90] J. R. Burch, E. M. Clarke, K. L. McMillan, D. L. Dill, and L. J. Hwang. Symbolic Model Checking: 10^20 States and Beyond. In *5th Annual IEEE Symposium on Logic in Computer Science, 4-7 June 1990, Philadelphia, Pennsylvania, USA*, LICS, pages 428 439. IEEE Computer Society, 1990.

[Ber00] G. Berry. The Foundations of Esterel. *Proof, language, and interaction: essays in honour of Robin Milner*, pages 425 454, 2000.

[Ber02] S. Berezin. *Model Checking and Theorem Proving: A Unified Framework*. PhD thesis, Carnegie Mellon University, Pittsburg, PA 15213, January 2002.

[BFPT06] B. Badban, M. Fränzle, J. Peleska, and T. Teige. Test Automation for Hybrid Systems. In *SOQUA '06: Proceedings of the 3rd Int. Workshop on Software Quality Assurance*, pages 14 21, New York, NY, USA, 2006. ACM Press.

[BGG88] A. Benveniste, B. Le Goff, and P. Le Guernic. Hybrid Dynamical Systems Theory and the Language 'SIGNAL'. Technical Report Publication Interne No 403, IRISA/INRIA, April 1988.

[BGHS04] S. Burmester, H. Giese, M. Hirsch, and D. Schilling. Incremental Design and Formal Verification with UML/RT in the FUJABA Real-Time Tool Suite . In *SVERTS2004, Satellite Event of the 7th International Conference on the Unified Modeling Language, UML2004*, pages 1 20, 2004.

[BGL+00] S. Bensalem, V. Ganesh, Y. Lakhnech, C. Muñoz, S. Owre, H. Rueß, J. Rushby, V. Rusu, H. Saïdi, N. Shankar, E. Singerman, and A. Tiwari. An Overview of SAL. In C. Michael Holloway, editor, *LFM 2000: Fifth NASA Langley Formal Methods Workshop*, pages 187 196, Hampton, VA, 2000.

[BGO04] S. Burmester, H. Giese, and O. Oberschelp. Hybrid UML Components for the Design of Complex Self-Optimizing Mechatronic Systems. In *ICINCO'04*, pages 222 229. INSTICC Press, 2004.

[Bis05] S. Bisanz. Executable HybridUML Semantics. A Transformation Definition. Dissertation, 2005.

[BK08] C. Baier and J.-P. Katoen. *Principles of Model Checking*. The MIT Press, Cambridge, MA, 2008.

[BKK02] A. Borshchev, Y. Karpov, and V. Kharitonov. Distributed Simulation of Hybrid Systems with AnyLogic and HLA. *Future Generation Computer Systems*, 18(6):829 839, May 2002.

[BL95] A. Bouajjani and Y. Lakhnech. Temporal Logic + Timed Automata: Expressiveness and Decidability. In *CONCUR*, volume 962 of *LNCS*, pages 531 545. Springer, 1995.

[BLL+05] C. Brooks, E.A. Lee, X. Liu, S. Neuendorffer, Y. Zhao, and H. Zheng. PTOLEMY II - Heterogeneous Concurrent Modeling and Design in JAVA, July 2005. Memorandum UCB/ERL/ M05/21, Department of Electrical Engineering and Computer Sciences, University of Califonia at Berkeley.

[BLL+06] H. Buchholz, A. Landsmann, P. Luksch, G. Riedewald, and E. Tetzner. Verification in High-Performance and Distributed Computing with Hybrid Systems for Symbolic Simulation. Preprint CS-05-06, University of Rostock, IEF, 2006.

[BLLT07a] H. Buchholz, A. Landsmann, P. Luksch, and E. Tetzner. Verifikation Zeitkritischer Eigenschaften Paralleler Programme. In *PARS'07*, pages 156 165, Hamburg-Harburg, Germany, Dezember 2007.

[BLLT07b] H. Buchholz, A. Landsmann, P. Luksch, and E. Tetzner. Verifikation Zeitkritischer Eigenschaften Paralleler Programme. RIB 31, Universität Rostock, Juni 2007.

[BLN03] D. Beyer, C. Lewerentz, and A. Noack. Rabbit: A Tool for BDD-Based Verification of Real-Time Systems. In *CAV*, volume 2725 of *LNCS*, pages 122 125, 2003.

[BM97] M. S. Branicky and S. E. Mattson. Simulation of Hybrid Systems in Omola/Omsim. In *Proc. of the 7th Symposium on Computer Aided Control Systems Design, CACSD'97*, 1997.

[BM05a] J. A. Bergstra and C. A. Middleburg. Process Algebra for Hybrid Systems. *Theoreoretical Computer Science*, 335(2-3):215 280, 2005.

[BM05b] T. Brihaye and C. Michaux. On the Expressiveness and Decidability of O-minimal Hybrid Systems. *Journal of Complexity*, 21(4):447 478, 2005.

[BMN00] P. Bellini, R. Mattolini, and P. Nesi. Temporal Logics for Real-Time System Specification. *ACM Computing Surveys*, 32(1):12 42, 2000.

[Bou08] P. Bouyer. Model-Checking Timed Temporal Logics. In Carlos Areces and Stéphane Demri, editors, *4th Workshop on Methods for Modalities (M4M-5)*, Electronic Notes in Theoretical Computer Science, Cachan, France, 2008. Elsevier Science Publishers. To appear.

[BR01] Dirk Beyer and Heinrich Rust. Cottbus Timed Automata: Formal Definition and Semantics. In C. Rattray, M. Sveda, and J. Rozenblit, editors, *Proceedings of the 2nd IEEE/IFIP Joint Workshop on Formal Specifications of Computer-Based Systems (FSCBS 2001, Washington, D.C., April 20)*, pages 75 87, Stirling, 2001.

[Bra00] R. Brauch. Graphische Repräsentation von Hybriden Systemen für die Sprache MODEL-HS, Dezember 2000. Diplomarbeit.

[BRJ98] G. Booch, J. Rumbaugh, and I. Jacobson. *The Unified Modeling Language User Guide*. Addison-Wesley, 1998.

[Bry86] R. E. Bryant. Graph-Based Algorithms for Boolean Function Manipulation. *IEEE Trans. Computers*, 35(8):677 691, 1986.

[BS97] S. Bornot and J. Sifakis. Relating Time Progress and Deadlines in Hybrid Systems. In *HART '97*, pages 286 300, London, UK, 1997. Springer-Verlag.

[BS02] M. Baldamus and T. Stauner. Modifying Esterel Concepts to Model Hybrid Systems. *Electronic Notes in Theoretical Computer Science*, 65(5), 2002.

[BSW02] R.-J. Back, C. Cerschi Seceleanu, and J. Westerholm. Symbolic Simulation of Hybrid Systems. In *APSEC '02*, pages 147 155, Washington, DC, USA, 2002. IEEE Computer Society.

[BW90] J. C. M. Baeten and W. P. Weijland. *Process Algebra*. Cambridge University Press, New York, NY, USA, 1990.

[BW01] A. Burns and A.J. Wellings. *Real-time Systems and Programming Languages. Third Edition*. Addison Wesley Longmain, Wokingham, 2001.

[CBP+04] L. Carloni, M. D. D. Benedetto, R. Passerone, A. Pinto, and A. Sangiovanni-Vincentelli. Modeling Techniques, Programming Languages and Design Toolsets for Hybrid Systems, July 2004. Project IST-2001-38314 COLUMBS - Design of Embedded Controllers for Safety Critical Systems, WPHS: Hybrid System Modeling, Version: 0.2, Deliverable number: DHS4-5-6.

[CBRZ01] E. M. Clarke, A. Biere, R. Raimi, and Y. Zhu. Bounded Model Checking Using Satisfiability Solving. *Formal Methods in System Design*, 19(1):7 34, 2001.

[CCG+02] A. Cimatti, E. M. Clarke, E. Giunchiglia, F. Giunchiglia, M. Pistore, M. Roveri, R. Sebastiani, and A. Tacchella. NuSMV 2: An OpenSource Tool for Symbolic Model Checking. In *CAV'02*, volume 2404 of *LNCS*, pages 359 364. Springer, 2002.

[CCK04] P. Chauhan, E. M. Clarke, and D. Kroening. A SAT-based Algorithm for Reparameterization in Symbolic Simulation. In *DAC '04*, pages 524 529, New York, NY, USA, 2004. ACM.

[CCK07] P. Chauhan, E. Clarke, and D. Kroening. Experiments with SAT-Based Symbolic Simulation Using Reparameterization in the Abstraction Refinement Framework. Technical report, Defense Technical Information Center OAI-PMH Repository (United States), 2007.

[CDE+02] M. Clavel, F. Durán, S. Eker, P. Lincoln, N. Martí-Oliet, J. Meseguer, and J. F. Quesada. Maude: Specification and Programming in Rewriting Logic. *Theoretical Computer Science*, 285(2):187 243, 2002.

[CE82] E. M. Clarke and E. A. Emerson. Design and Synthesis of Synchronization Skeletons Using Branching-Time Temporal Logic. In *Logic of Programs, Workshop, Yorktown Heights, New York, May 1981*, volume 131 of *LNCS*, pages 52 71. Springer, 1982.

[Cel77] F. E. Cellier. Combined Continuous/Discrete System Simulation Languages: Usefulness, Experiences and Future Development. *SIGSIM Simul. Dig.*, 9(1):18 21, 1977.

[CFH+03] E. M. Clarke, A. Fehnker, Z. Han, B. H. Krogh, J. Ouaknine, O. Stursberg, and M. Theobald. Abstraction and Counterexample-Guided Refinement in Model Checking of Hybrid Systems. *Foundation of Computer Science*, 14(4):583 604, 2003.

[CG87] E. M. Clarke and O. Grumberg. Avoiding the State Explosion Problem in Temporal Logic Model Checking. In *PODC '87: Proceedings of the sixth annual ACM Symposium on Principles of distributed computing*, pages 294 303, New York, NY, USA, 1987. ACM.

[CGP99] E. M. Clarke, O. Grumberg, and D. A. Peled. *Model Checking*. The MIT Press, Cambridge, Massachusetts, 1 edition, 1999.

[CGP+02] A. Cimatti, E. Giunchiglia, M. Pistore, M. Roveri, R. Sebastiani, and A. Tacchella. Integrating BDD-Based and SAT-Based Symbolic Model Checking. In *FroCoS '02*, pages 49 56, London, UK, 2002. Springer-Verlag.

[CI07] A. Cornell and D. Ionescu. *Real-Time Systems: Modeling, Design, and Applications*. World Scientific Publishing Company, 1 edition, 2007.

[CKOS04] E. M. Clarke, D. Kroening, J. Ouaknine, and O. Strichman. Completeness and Complexity of Bounded Model Checking. In *VMCAI*, volume 2937 of *LNCS*, pages 85 96. Springer, 2004.

[CL99] C.G. Cassandras and S. Lafortune. *Introduction to Discrete Event Systems*. Springer Science+Business Media Inc., New York, USA, 1999.

[CPPSV06] L. Carloni, R. Passerone, A. Pinto, and A. Sangiovanni-Vincentelli. Languages and Tools for Hybrid Systems Design. *Foundation and Trends in Electronic Design Automation*, 1(1/2), 2006.

[CR04] P.J.L. Cuijpers and M.A. Reniers. Action and Predicate Safety of Hybrid Processes. Technical Report CSR-04-10, Department of Computer Science, Eindhoven University of Technology, Eindhoven, The Netherlands, 2004.

[CR05] P.J.L. Cuijpers and M.A. Reniers. Hybrid Process Algebra. *Journal of Logic and Algebraic Programming*, 62(2):191 245, 2005.

[dAHS02] L. de Alfaro, T. A. Henzinger, and M. Stoelinga. Timed Interfaces. In *EMSOFT*, volume 2491 of *LNCS*, pages 108 122. Springer, 2002.

[DC96] D. Damon and E. Christen. Introduction to VHDL-AMS. 1. Structural and discrete time concepts. In *Proceedings of the IEEE International Symposium on Computer-Aided Control System Design*, pages 264 269. IEEE, 1996.

[DGS98] A. Deshpande, A. Göllü, and L. Semenzato. The SHIFT Programming Language and Run-time System for Dynamic Networks of Hybrid Automata. *IEEE transactions on automatic control*, 43(4):584 587, 1998.

[DGV97] A. Deshpande, A. Göllü, and P. Varaiya. SHIFT: A Formalism and a Programming Language for Dynamic Networks of Hybrid Automata. In *Hybrid Systems*, volume 1273 of *Lecture Notes in Computer Science*, pages 113 133. Springer, 1997.

[DHR+07] M.B. Dwyer, J. Hatcliff, R. Robby, C.S. Pasareanu, and W. Visser. Formal Software Analysis Emerging Trends in Software Model Checking. In *FOSE '07: 2007 Future of Software Engineering*, pages 120 136, Washington, DC, USA, 2007. IEEE Computer Society.

[DJPV02] W. Damm, B. Josko, A. Pnueli, and A. Votintseva. Understanding UML: A Formal Semantics of Concurrency and Communication in Real-Time UML. In Frank S. de Boer, Marcello M. Bonsangue, Susanne Graf, and Willem P. de Roever, editors, *FMCO*, volume 2852 of *LNCS*, pages 71 98. Springer, 2002.

[DLN+99] R. Djenidi, C. Lavarenne, R. Nikoukhah, Y. Sorel, and S. Steer. From Hybrid System Simulation to Real-Time Implementation, 1999. In 11th European Simulation Symposium and Exhibition, Erlangen-Nürnberg, Oct. 1999.

[dMRS02] L. de Moura, H. Rueß, and M. Sorea. Lazy Theorem Proving for Bounded Model Checking over Infinite Domains. In *CADE'02*, volume 2392 of *LNCS*, pages 438 455. Springer-Verlag, 2002.

[DP99] G. Delzanno and A. Podelski. Model Checking in CLP. In *TACAS '99*, pages 223 239, London, UK, 1999. Springer-Verlag.

[DP01] G. Delzanno and A. Podelski. Constraint-based Deductive Model Checking. *Software Tools for Technology Transfer (STTT)*, 3(3), 2001.

[EBB+99] H. Elmqvist, B. Bachmann, F. Boudard, J. Broenink, D. Brück, T. Ernst, R. Franke, P. Fritzson, A. Jeandel, P. Grozman, K. Juslin, D. Kågedal, M. Klose, N. Loubére, S.-E. Mattson, P. Mosterman, H. Nilsson, M. Otter, P. Sahlin, A. Schneider, H. Tummescheit, and H. Vangheluwe. Modelica - a Unified Object-Oriented Language for Physical Systems Modeling: Tutorial and Rationale. Report, The Modelica Design Group. <http://www.modelica.org/>, 1999.

[ECO05] H. Elmquist, F. Cellier, and M. Otter. Object-Oriented Modeling of Hybrid Systems. In *Proceedings of the 9th Annual SCS European Multiconference*

on Simulation, ESM '95, SCS Publication, pages 31 39, Delft, The Netherlands, 2005.

[EFH08] A. Eggers, M. Fränzle, and C. Herde. SAT Modulo ODE: A Direct SAT approach to Hybrid Systems. Reports of SFB/TR 14 AVACS 37, SFB/TR 14 AVACS, April 2008. ISSN: 1860-9821.

[EHS97] J. Ellsberger, D. Hogrefe, and A. Sarma. *SDL-Formal Object-oriented Language for Communicating Systems*. Prentice Hall, London, 1997.

[Eic02] C. Eichholz. Rahmenwerk zur Zustandsbasierten Klassischen und Symbolischen Simulation Hybrider Systeme - Verifikation und Simulation Hybrider Systeme im Vergleich, März 2002. Diplomarbeit.

[EKKT08] A. Eggers, N. Kalinnik, S. Kupferschmid, and T. Teige. Challenges in Constraint-based Analysis of Hybrid Systems. In *Proceedings of the Annual ERCIM Workshop on Constraint Solving and Constraint Logic Programming (CSCLP 2008)*, June 2008.

[EKRNS00] T. Ernst, C. Klein-Robbenhaar, A. Nordwig, and T. Schrag. Modellierung und Simulation Hybrider Systeme mit Smile. *Informatik - Forschung und Entwicklung*, 15(1):33 50, 2000.

[Elm93] H. Elmqvist. Dymola - User's Manual. Technical Report Version 2.0, DynaSim AB, Research Park Ideon, Lund, SwedenFrauenhofer, 1993.

[ENNG+05] T. Ernst, A. Nordwig, C. Nytsch-Geusen, P. Schneider, P. Schwarz, M. Vetter, C. Wittwer, A. Holm, T. Nouidui, J. Leopold, G. Schmidt, U. Doll, and A. Mattes. MOSILAB: Development of a Modelica Based Generic Simulation Tool Supporting Modal Structural Dynamics. In G. Schmitz, editor, *Modelica 2005, 4th International Modelica Conference*, Proceedings, pages 527 534. Hamburg University of Technology, Hamburg-Harburg, March 2005.

[ENNG+06] T. Ernst, A. Nordwig, C. Nytsch-Geusen, C. Clauß, and A. Schneider. MOSILA Modellbeschreibungssprache - Modellierung und Simulation komplexer technischer Systeme - Spezifikation. Technical Report Version 2.0, Frauenhofer Institut FIRST, Berlin, 2006.

[EP02] C. Eisner and D. Peled. Comparing Symbolic and Explicit Model Checking of a Software System. In *Model Checking of Software: 9th Int. SPIN Workshop*, number 2318 in LNCS, pages 79 82. Springer, 2002.

[FA97] T. Frühwirth and S. Abdennadher. *Constraint-Programmierung*. Springer Verlag, Berlin, 1997.

[FGG+05] I. Fliege, A. Geraldy, R. Gotzhein, T. Kuhn, and C. Webel. Developing Safety-Critical Real-Time Systems with SDL Design Patterns and Components. *Computer Network*, 49(5):689 706, 2005.

[FH05] M. Fränzle and C. Herde. Efficient Proof Engines for Bounded Model Checking of Hybrid Systems. *Electronic Notes of Theoretical Computer Science*, 133:119 137, 2005.

[FH07] M. Fränzle and C. Herde. HySAT: An Efficient Proof Engine for Bounded Model Checking of Hybrid Systems. *Formal Methods in System Design*, 30(3):179 198, 2007.

[Fla03] Cormac Flanagan. Automatic Software Model Checking Using CLP. In *ESOP'03*, pages 189 203, 2003.

[FNORC08] G. Faure, R. Nieuwenhuis, A. Oliveras, and E. Rodríguez-Carbonell. SAT Modulo the Theory of Linear Arithmetic: Exact, Inexact and Commercial Solvers. In *SAT*, volume 4996 of *LNCS*, pages 77 90. Springer, 2008.

[FNW98] V. Friesen, A. Nordwig, and M. Weber. Object-Oriented Specification of Hybrid Systems Using UML and ZimOO. In *ZUM*, pages 328 346, 1998.

[Fok00] W. Fokkink. *Introduction to Process Algebra*. Springer, Berlin, 2000.

[Fre05] G. Frehse. PHAVer: Algorithmic Verification of Hybrid Systems Past HyTech. In *HSCC 2005*, volume 3414 of *Lecture Notes in Computer Science*, pages 258 273. Springer, 2005.

[FT00] J.-M. Flaus and L. Thévenon. The YASMA Language for Representing Systems, 2000. Technical Report, Laboratoire d'Automatique de Grenoble, France.

[Gar02] D. Garriou. Symbolic Simulation of Synchronous Programs. *Theoretical Computer Science*, 65(5), 2002.

[GB03] H. Giese and S. Burmester. Real-Time Statechart Semantics. Technical Report tr-ri-03-239, Lehrstuhl für Softwaretechnik, Universität Paderborn, Paderborn, Germany, 2003.

[GG07] R. Grammes and R. Gotzhein. SDL Profiles - Formal Semantics and Tool Support. In *Fundamental Approaches to Software Engineering*, volume 4422 of *LNCS*, pages 200 214. Springer, 2007.

[GGP03] U. Glässer, R. Gotzhein, and A. Prinz. The Formal Semantics of SDL-2000: Status and Perspectives. *Computer Networks*, 42(3):343 358, 2003.

[GH04] S. Graf and J. Hooman. Correct Development of Embedded Systems. In *EWSA*, volume 3047 of *LNCS*, pages 241 249. Springer, 2004.

[GHK00] G. Graw, P. Herrmann, and H. Krumm. Verification of UML-Based Real-Time System Designs by Means of cTLA. In *ISORC '00*, pages 86 95, Washington, DC, USA, 2000. IEEE Computer Society.

[GHKR98] S. Di Gennaro, C. Horn, S. R. Kulkarni, and P. J. Ramadge. Reduction of Timed Hybrid Systems. *Discrete Event Dynamic Systems*, 8(4):343 351, 1998.

[GJS96] V. Gupta, R. Jagadeesan, and V. Saraswat. Hybrid cc, Hybrid Automata and Program Verification. In *Proc. of the DIMACS/SYCON workshop on Hybrid systems III : verification and control*, pages 52 63, Secaucus, NJ, USA, 1996. Springer-Verlag New York, Inc.

[GJSB95] V. Gupta, R. Jagadeesan, V.A. Saraswat, and D.G. Bobrow. Programming in Hybrid Constraint Languages. In *Hybrid Systems II*, volume 999 of *LNCS*, pages 226 251. Springer, 1995.

[GLL99] A. Girault, B. Lee, and E.A. Lee. Hierarchical Finite State Machines with Multiple Concurrency Models. *IEEE Transactions on Computer-Aided Design*, 18(6):742 760, June 1999.

[GNRR93] R. L. Grossman, A. Nerode, A. P. Ravn, and H. Rischel, editors. *Hybrid Systems*, volume 736 of *LNCS*. Springer, 1993.

[GNRZ07] S. Ghilardi, E. Nicolini, S. Ranise, and D. Zucchelli. Combination Methods for Satisfiability and Model-Checking of Infinite-State Systems. In *Automated Deduction - CADE-21*, volume 4603 of *LNCS*, pages 362 378. Springer, 2007.

[GOO04] S. Graf, I. Ober, and I. Ober. Model checking of UML Models via a Mapping to Communicating Extended Timed Automata. In Susanne Graf and Laurent Mounier, editors, *SPIN'04 Workshop, Barcelona, Spain*, volume 2989 of *LNCS*. Springer, 2004.

[Gor05] T. Gornig. Abstraktion und Verfeinerung in Sprachen für die Symbolische Simulation Hybrider Systeme. Studienarbeit, September 2005.

[Gra08] S. Graf. Omega Correct Development of Real Time Embedded Systems. *SoSyM, int. Journal on Software & Systems Modelling*, 7(2), 2008.

[GS98] R. Grosu and T. Stauner. Visual Description of Hybrid Systems. In *Workshop On Real Time Programming (WRTP'98)*. Elsevier Science Ltd., 1998.

[GS02] R. Grosu and T. Stauner. Modular and Visual Specification of Hybrid Systems: An Introduction to HyCharts. *Formal Methods System Design*, 21(1):5 38, 2002.

[Ham05] Y. Hammal. A Formal Semantics of UML StateCharts by Means of Timed Petri Nets. In *FORTE 2005*, number 3731 in LNCS, pages 38 52, 2005.

[Har87] D. Harel. Statecharts: A Visual Formalism for Complex Systems. *Science of Computer Programming*, 8(3):231 274, 1987.

[HEFT08] C. Herde, A. Eggers, M. Fränzle, and T. Teige. Analysis of Hybrid Systems using HySAT. In *ICONS 2008*, pages 196 201. IEEE Computer Society, 2008.

[Hen96] T.A. Henzinger. The Theory of Hybrid Automata. In *11th Annual Symposium on LICS*, LICS, pages 278 292. IEEE Computer Society Press, 1996.

[Hen00] T. A. Henzinger. Masaccio: A Formal Model for Embedded Components. In *IFIP TCS*, volume 1872 of *Lecture Notes in Computer Science*, pages 549 563. Springer, 2000.

[HHWT95] T.A. Henzinger, P.-H. Ho, and H. Wong-Toi. A User Guide to HyTech. In *TACAS '95: Proceedings of the First International Workshop on Tools and Algorithms for Construction and Analysis of Systems*, pages 41 71, London, UK, 1995. Springer.

[HK97] P. Herrmann and H. Krumm. Kompositionale Constraints hybrider Systeme. In E. Schnieder und D. Abel, editor, *Entwurf komplexer Automatisierungssysteme*, Tagungsband zur 5. Fachtagung, pages 243 264, Braunschweig, 1997.

[HKPV98] T. A. Henzinger, P. W. Kopke, A. Puri, and P. Varaiya. What's decidable about hybrid automata? *Computer and System Sciences*, 57(1):94 124, 1998.

[HM06] T.A. Henzinger and S. Matic. An Interface Algebra for Real-Time Components. In *RTAS '06: Proceedings of the 12th IEEE Real-Time and Embedded Technology and Applications Symposium*, pages 253 266, Washington, DC, USA, 2006. IEEE Computer Society.

[HMP01] T. Henzinger, M. Minea, and V. Prabhu. Assume-Guarantee Reasoning for Hierarchical Hybrid Systems. In *HSCC 2001*, number 2034 in LNCS, pages 275 290. Springer Verlag, 2001.

[HNSY94] T. A. Henzinger, X. Nicollin, J. Sifakis, and S. Yovine. Symbolic Model Checking for Real-Time Systems. *Information and Computation*, 111(2):193 244, 1994.

[HPSS87] D. Harel, A. Pnueli, J. P. Schmidt, and R. Sherman. On the Formal Semantics of Statecharts. In *LICS*, pages 54 64, Ithaca, NY, 1987.

[HR00] M.R.A. Huth and M. Ryan. *Logic in Computer Science: Modelling and Reasoning about Systems*. Cambridge University Press, New York, NY, USA, 2000.

[HS88] M. Höhfeld and G. Smolka. Definite Relations Over Constraint Languages. LILOG Report 53, IWBS, IBM Deutschland, Stuttgart, 1988.

[HW04] T. J. Hickey and D. K. Wittenberg. Rigorous Modeling of Hybrid Systems Using Interval Arithmetic Constraints. In *HSCC*, volume 2993 of *LNCS*, pages 402 416. Springer, 2004.

[HW07] P. Hofstedt and A. Wolf. *Einführung in die Constraint-Programmierung - Grundlagen, Methoden, Sprachen, Anwendungen*. Springer, Berlin, 2007.

[IEE97] IEEE. Standard VHDL Analog and Mixed-Signal Extensions, 1997. Technical Report IEEE 1076.1.

[Jif94] H. Jifeng. From CSP to Hybrid Systems. *A Classical Mind: Essays in Honour of C. A. R. Hoare*, pages 171 189, 1994.

[JKSC05] H. Jain, D. Kroening, N. Sharygina, and E. Clarke. Word Level Predicate Abstraction and Refinement for Verifying RTL Verilog. In *Proceedings of DAC 2005*, pages 445 450, 2005.

[JKWC07] S. K. Jha, B. H. Krogh, J. E. Weimer, and E. M. Clarke. Reachability for Linear Hybrid Automata Using Iterative Relaxation Abstraction. In *HSCC'07*, volume 4416 of *LNCS*, pages 287 300. Springer, 2007.

[JL87] J. Jaffer and J.L. Lassez. Constraint Logic Programming. In *Proc of the 14th ACM Symposium on Principles of Programming Languages POPL'87*, pages 111 119, München, 1987.

[JM94] J. Jaffer and M.J. Maher. Constraint Logic Programming: A Survey. *Journal of Logic Programming*, 20:503 581, 1994.

[KB00] M. Kokar and K. Baclawski. Modeling Combined Time- and Event-Driven Dynamic Systems. In *Ninth OOPSLA Workshop on Behavioral Semantics*, pages 112 129, 2000.

[Ket92] D.L. Kettenis. COSMOS: A Simulation Language for Continuous, Discrete and Combined Models. *Simulation*, 58(1):32 41, 1992.

[Ket94] D. Kettenis. *Issues of Parallelization in Implementation of the Combined Simulation Language COSMOS*. PhD thesis, Delft University of Technology, 1994.

[KFS95] M. Kloas, V. Friesen, and M. Simons. Smile - A Simulation Environment for Energy Systems. *Systems Analysis Modelling Simulation*, 18-19:503 506, 1995.

[Kot97] M.P. Kottmann. *Komponentenorientierte Modellierung und Simulation kombinierter Systeme*. PhD thesis, Eidgenössische technische Hochschule Zürich, 1997.

[KP92] Y. Kesten and A. Pnueli. Timed and Hybrid Statecharts and Their Textual Representation. In *FTRTFT*, volume 571 of *Lecture Notes in Computer Science*, pages 591 620. Springer, 1992.

[Kri06] T. Krilavičius. *Hybrid Techniques for Hybrid Systems*. PhD thesis, University of Twente, The Netherlands, 2006.

[KRS07] F. Klaedtke, S. Ratschan, and Z. She. Language-Based Abstraction Refinement for Hybrid System Verification. In *VMCAI*, volume 4349 of *LNCS*, pages 151 166. Springer, 2007.

[KSPL06] F. Kratz, O. Sokolsky, G. J. Pappas, and I. Lee. R-Charon, a Modeling Language for Reconfigurable Hybrid Systems. In *HSCC*, volume 3927 of *Lecture Notes in Computer Science*, pages 392 406. Springer, 2006.

[KV04] M. V. Korovina and N. Vorobjov. Pfaffian Hybrid Systems. In *CSL: 13th Annual Conference of the EACSL*, pages 430 441. Springer, 2004.

[LAB07] X. Li, S. J. Aanand, and L. Bu. Towards an Efficient Path-Oriented Tool for Bounded Reachability Analysis of Linear Hybrid Systems using Linear Programming. *Theoretical Computer Science*, 174(3):57 70, 2007.

[Lap04] P.A. Laplante. *Real-time Systems Design and Analysis*. Willey-IEEE Press, 3 edition, 2004.

[LC05] J. Lee and S. Chi. Using Symbolic DEVS Simulation to Generate Optimal Traffic Signal Timings. *Simulation*, 81(2):153 170, 2005.

[Loh05] M. Lohrey. Model-Checking Hierarchical Structures. *LICS*, 00:168 177, 2005.

[LPS00] G. Lafferriere, G.J. Pappas, and S. Sastry. O-Minimal Hybrid Systems. *Math. Control Signals Systems*, 13:1 21, 2000.

[LPY97] K.G. Larsen, P. Pettersson, and W. Yi. UPPAAL: Status and Developments. In Orna Grumberg, editor, *CAV97*, number 1254 in LNCS, pages 456 459. Springer Verlag, Jun 1997.

[LR01] R. Lämmel and G. Riedewald. Prological Language Processing. In Mark G. van den Brand and Didier Parigot, editors, *LDTA'01*, volume 44 of *ENTCS*. Elsevier Science, April 2001.

[LSV03] N. Lynch, R. Segala, and F. Vaandrager. Hybrid I/O automata. *Information and Computation*, 185(1):105 157, 2003.

[LTR03] A. Lantsmann, E. Tetzner, and G. Riedewald. Ubiquitäre Studienplanung aus Sicht der Konsistenz und Sicherheit, 2003. ISSN 0233-0784.

[LvdBC00] G. Lüttgen, M. von der Beeck, and R. Cleaveland. A Compositional Approach to Statecharts Semantics. In *SIGSOFT FSE*, pages 120 129, 2000.

[LZ05] E. A. Lee and H. Zheng. Operational Semantics of Hybrid Systems. In *HSCC*, volume 3414 of *LNCS*, pages 25 53. Springer, 2005.

[LZ06] E. A. Lee and H. Zheng. HyVisual: A Hybrid System Modeling Framework Based on Ptolemy II. In *IFAC Conference on Analysis and Design of Hybrid Systems*, January 2006.

[Mar07] P. Marwedel. *Eingebettete Systeme*. Springer Verlag, Berlin, 1 edition, 2007.

[Mau96] C. Mauras. Symbolic Simulation of Interpreted Automata. 3rd Workshop on Synchronous Languages, Dezember 1996. Dagstuhl (Germany).

[McM93] K. McMillan. *Symbolic Model Checking*. PhD thesis, CMU, Kluwer Academic, 1993.

[Mer01] S. Merz. Model Checking: A Tutorial Overview. In *MOVEP '00: 4th Summer School on Modeling and Verification of Parallel Processes*, number 2067 in LNCS, pages 3 38, London, UK, 2001. Springer-Verlag.

[Mil99] R. Milner. *Communicating and Mobile Systems: the Pi-Calculus*. Cambridge University Press, June 1999.

[Mis95] L.R. Missiaen. ECSIM: Discrete Event Simulation using Event Calculus. *Information Systems Division*, 1995. SHAPE Technical Centre.

[Mit07] Sayan Mitra. *A Verification Framework for Hybrid Systems*. PhD thesis, Massachusetts Institute of Technology, Cambridge, MA 02139, September 2007.

[MLAH99] J. B. Møller, J. Lichtenberg, H. R. Andersen, and H. Hulgaard. Fully Symbolic Model Checking of Timed Systems using Difference Decision Diagrams. *Electronic Notes Theoretical Computer Science*, 23(2), 1999.

[MN95] O. Müller and T. Nipkow. Combining Model Checking and Deduction for I/O-Automata. In *TACAS*, volume 1019 of *LNCS*, pages 1 16. Springer, 1995.

[MOE98] P. Mosterman, M. Otter, and H. Elmqvist. Modeling Petri Nets as Local Constraint Equations for Hybrid Systems Using Modelica. In Mohammad S. Obaidat Franco Davoli Danny DeMarinis, editor, *Proceedings of the Summer Computer Simulation Conference-98*, pages 314 319. The Society for Computer Simulation International, 1998.

[MOE99] S.E. Mattsson, M. Otter, and H. Elmqvist. Modelica Hybrid Modeling and Efficient Simulation. In IEEE, editor, *Proc. of the 38th IEEE Conference on Decision and Control, CDC'99*, pages 3502 3507, 1999.

[Möl92] D. Möller. *Modellbildung, Simulation und Identifikation dynamischer Systeme*. Springer, Berlin, 1992.

[Mos99] P.J. Mosterman. An Overview of Hybrid Simulation Phenomena and their Support by Simulation Packages. In *HSCC'99*, number 1569 in LNCS, pages 178 192. Springer Verlag, 1999.

[Mos02] P.J. Mosterman. HyBrSim - A Modeling and Simulation Environment for Hybrid Bond Graphs. *Journal of Systems and Control Engineering*, 216:35 46, 2002.

[MP92] Z. Manna and A. Pnueli. *The Temporal Logic of Reactive and Concurrent Systems - Specification*. Springer, New York, 1992.

[MP95] Z. Manna and A. Pnueli. *Temporal Verification of Reactive Systems - Safety*. Springer, New York, 1995.

[MRC05] K.L. Man, M.A. Reniers, and P.J.L. Cuijpers. Case Studies in the Hybrid Process Algebra HyPA. *Int. Journal of Software Engineering anf Knowledge Engineering*, 15(2):299 305, 2005.

[MT99] C. Mauras and R. Thoraval. A propos de la Vérification de Programmes Synchrones et de l'Analyse de Programmes Logiques aves Contraintes. In *JFPLC*, pages 39 54. Hermes, 1999.

[MT01] A. Mitschele-Thiel. *System Engineering with SDL - Developing Performance - Critical communication Systems*. WILEY, West Sussex, Po191UD, England, 2001.

[NC94] S. Narain and R. Chadha. Symbolic Discrete - Event Simulation. In P.R. Kumar and P. Varaiya, editors, *Discrete Event Systems, Manufacturing Systems and Communication Networks*, number 73 in IMA volumes in mathematics and its applications, pages 201 224. Springer, 1994.

[NGEN+06] C. Nytsch-Geusen, T. Ernst, A. Nordwig, P. Schwarz, P. Schneider, M. Vetter, C. Wittwer, A. Holm, T. Nouidui, J. Leopold, G. Schmidt, and A. Mattes. Advanced Modeling and Simulation Techniques in MOSILAB: A System Development Case Study. In C. Kral, editor, *Modelica 2006, 5th International Modelica Conference*, Proceedings, pages 63 72. Vienna, Austria, Modelica Association, Sep 2006.

[Nie02] L. P. Nieto. *Verification of Parallel Programs with the Owicki-Gries and Rely-Guarantee Methods in Isabelle/HOL*. PhD thesis, Technische Universität München, 2002.

[Nik06] R. Nikoukhah. Modeling hybrid systems in SCICOS: a case study. In *MIC'06: Proceedings of the 25th IASTED international conference on Modeling, identification, and control*, pages 315 319, Anaheim, CA, USA, 2006. ACTA Press.

[NOT04] R. Nieuwenhuis, A. Oliveras, and C. Tinelli. Abstract DPLL and Abstract DPLL Modulo Theories. In *LPAR'04*, volume 3452 of *Lecture Notes in Computer Science*, pages 36 50. Springer, 2004.

[NPW02] T. Nipkow, L.C. Paulson, and M. Wenzel. *Isabelle/HOL A Proof Assistant for Higher-Order Logic*, volume 2283 of *LNCS*. Springer, 2002.

[NS06] R. Nikoukhah and S. Steer. SCICOS A Dynamic System Builder and Simulator User's Guide - Version 1.0. Technical report, INRIA a CCSD electronic archive server based on P.A.O.L [http://hal.inria.fr/oai/oai.php] (France), 2006.

[NT08] M. Niqui and O. Tveretina. Modular Development of Hybrid Systems for Verification in Coq. In *HSCC'08*, LNCS, St. Louis, MO, USA, April 2008. Springer-Verlag. to appear.

[OEM99] M. Otter, H. Elmqvist, and S.E. Mattsson. Hybrid Modeling in Modelica based on the Synchronous Data Flow Principle. In *Proc. of the IEEE International Symposium on Computer Aided Control System Design, CACSD'99*, pages 151 157, 1999.

[ÖM04] P. C. Ölveczky and J. Meseguer. Specification and Analysis of Real-Time Systems Using Real-Time Maude. In *Proc. of Fundamental Approaches to Software Engineering (FASE)*, number 2984 in LNCS, pages 354 358. Springer Verlag, 2004.

[ORSSC98] S. Owre, J. Rushby, N. Shankar, and D. Stringer-Calvert. PVS: An Experience Report. In D. Hutter, W. Stephan, P. Traverso, and M. Ullman, editors, *Applied Formal Methods FM-Trends 98*, volume 1641 of *LNCS*, pages 338 345, Boppard, Germany, 1998.

[PG97] E. Pontelli and G. Gupta. A Constraint Based Approach for Specification and Verification of Real-time Systems. In *18th IEEE Real Time Systems Symposium*, RTSS'97. IEEE Computer Society, 1997.

[PLPD02] T. Pawletta, B. Lampe, S. Pawletta, and W. Drewelow. A DEVS-Based Approach for Modeling and Simulation of Hybrid Variable Structure Systems. In *Modeling, Anlysis, and Design of Hybrid Systems*, number 279 in Lecture Notes in Control and Information Sciences, pages 107 129, 2002.

[Por01] I. Porres. *Modeling and Analysing Software Behaviour in UML*. PhD thesis, Åbo Akademi University Turku, 2001.

[PQ08] A. Platzer and J.-D. Quesel. Logical Verification and Systematic Parametric Analysis in Train Control. In M. Egerstedt and B. Mishra, editors, *HSCC 2008, St. Louis, USA, Proceedings*, LNCS. Springer, 2008.

[PR07] A. Podelski and A. Rybalchenko. ARMC: The Logical Choice for Software Model Checking with Abstraction Refinement. In *PADL'07*, volume 4354 of *LNCS*, pages 245 259. Springer, 2007.

[Pri01] A. Prinz. *Formal Semantics for SDL*. PhD thesis, Humboldt-Universität zu Berlin, Mathematisch-Naturwissenschaftliche Fakultät II, 2001.

[PSS95] C. D. Pegden, R. P. Sadowski, and R. E. Shannon. *Introduction to Simulation Using SIMAN*. McGraw-Hill, Inc., New York, NY, USA, 1995.

[QS82] J.-P. Queille and J. Sifakis. Specification and Verification of Concurrent Systems in CESAR. In *Symposium on Programming, 5th Colloquium*, volume 137 of *LNCS*, pages 337 351. Springer, 1982.

[QX04] J. Qian and B. Xu. The Compositional Semantics of Timed Statecharts. In *IASTED'04*, pages 345 349. IASTED/ACTA Press, 2004.

[RG04] J. Romberg and C. Grimm. Refinement of Hybrid Systems: From Formal Models to Design Languages. In *Languages for system specification: Selected contributions on UML, systemC, system Verilog, mixed-signal systems, and property specification from FDL'03*, pages 315 330, Norwell, MA, USA, 2004. Kluwer Academic Publishers.

[Rie95] G. Riedewald. A Little Bit Modified Railroad Crossing. Preprint CS-05-95, ISSN 0944-5900, University Rostock, Department of Computer Science, 1995.

[RRS03] M. Rönkkö, A. P. Ravn, and K. Sere. Hybrid Action Systems. *Theoretical Computer Science*, 290(1):937 973, 2003.

[RS02] W. C. Rounds and H. Song. The Phi-Calculus - a Hybrid Extension of the Pi-Calculus to Embedded Systems. In *Proc. of 18th Workshop on the Mathematical Foundations of Programming Semantics*. New Orleans, 2002.

[RS03] W. C. Rounds and H. Song. The Phi-Calculus: A Language for Distributed Control of Reconfigurable Embedded Systems. In *HSCC 2003*, volume 2623 of *LNCS*, pages 435 449. Springer, 2003.

[RS07] S. Ratschan and Z. She. Safety Verification of Hybrid Systems by Constraint Propagation Based Abstraction Refinement. *ACM Transactions in Embedded Computing Systems*, 6(1), 2007.

[RSM06] R. Ramos, A. Sampaio, and A. Mota. Rigorous Development with UML-RT, 2006. 19th Brazilian Contest on Dissertations and Thesis (CTD'06), SBC.

[RU95] G. Riedewald and L. Urbina. Symbolische Simulation Hybrider Systeme in Constraint Logic Programming. Preprint CS-02-95/CS-03-95, University Rostock, Department of Computer Science, 1995.

[Rus01] John Rushby. Formal Verification of McMillan's Compositional Assume-Guarantee Rule. Technical report, SRI International Computer Science Laboratory, Menlo Park California, Sep 2001.

[Rus05] H. Rust. *Operational Semantics for Timed Systems: A Non-standard Approach to Uniform Modeling of Timed and Hybrid Systems*, volume 3456 of *LNCS*. Springer, 2005.

[RZD+07] A. Rettberg, M. C. Zanella, R. Dömer, A. Gerstlauer, and F.-J. Rammig, editors. *Embedded System Design: Topics, Techniques and Trends, IFIP TC10 Working Conference: International Embedded Systems Symposium (IESS), May 30 - June 1, 2007, Irvine, CA, USA*, volume 231 of *IFIP*. Springer, 2007.

[SASX05] S. Shankar, S. Asa, V. Sipos, and X. Xu. Reasoning about Real-Time Statecharts in the Presence of Semantic Variations. In *ASE '05*, pages 243 252, New York, NY, USA, 2005. ACM Press.

[SCR06] H. Song, K. J. Compton, and W. C. Rounds. SPHIN: A Model Checker for Reconfigurable Hybrid Systems based on SPIN. *Electronic Notes of Theoretical Computer Science*, 145:167 183, 2006.

[SdG92] R. W. Sierenberg and O. B. de Gans. Personal Prosim: A Fully Integrated Simulation Environment. In W. Krug and A. Lehmann, editors, *ESS-92: Proc. of the 1992 European Simulation Symposium*, pages 167 173. Society for Computer Simulation, San Diego, 1992.

[SGW94] B. Selic, G. Gullekson, and P. T. Ward. *Real-Time Object-Oriented Modeling*. John Wiley & Sons, April 1994.

[Sik05] M. Sikos. Fallstudie zur Beschreibung hybrider Systeme anhand der Sicherheitsanlagen eines Verkehrstunnels. Studienarbeit, Juni 2005.

[Sik06] M. Sikos. Übersetzung von Läufen der symbolischen Simulation hybrider Systeme in Ausdrücke temporaler Logik und Rücküberführung. Diplomarbeit, März 2006.

[SJG95] V. A. Saraswat, R. Jagadeesan, and V. Gupta. Default Timed Concurrent Constraint Programming. In *Conference Record of POPL '95: 22nd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 272 285, San Francisco, California, 1995.

[Smi00] G. Smith. *The Object-Z Specification Language*. Kluwer Academic, Hingham, USA, 2000.

[Sor02] M. Sorea. Bounded Model Checking for Timed Automata. *Electronic Notes in Theoretical Computer Science*, 68(5), 2002.

[SPP01] T. Stauner, A. Pretschner, and I. Péter. Approaching a Discrete-Continuous UML: Tool Support and Formalization. In *pUML*, volume 7 of *LNI*, pages 242 257. GI, 2001.

[SRKC00] B. I. Silva, K. Richeson, B. Krogh, and A. Chutinan. Modeling and Verifying Hybrid Dynamic Systems Using CheckMate. In *4th International Conference on Automation of Mixed Processes, September 2000*, pages 323 328, 2000.

[SS07] T. Schuele and K. Schneider. Bounded Model Checking of Infinite State Systems. *Formal Methods of System Design*, 30(1):51 81, 2007.

[Sta00] T. Stauner. Extending HyCharts with State-Invariants. In *GI workshop Rigorose Entwicklung software-intensiver Systeme*. Ludwig-Maximilians-Universität München, 2000.

[STG01] R. Sebastiani, A. Tomasi, and F. Giunchiglia. Model Checking Syllabi and Student Careers. In *TACAS 2001*, number 2031 in LNCS. Springer, 2001.

[SUM96] H.B. Sipma, T.E. Uribe, and Z. Manna. Deductive Model Checking. In *CAV'96*, volume 1102 of *LNCS*, pages 209 219. Springer, 1996.

[SY96] J. Sifakis and S. Yovine. Compositional Specification of Timed Systems (Extended Abstract). In *STACS*, volume 1046 of *LNCS*, pages 347 359. Springer, 1996.

[Tay93] J. H. Taylor. Toward a Modeling Language Standard for Hybrid Dynamical Systems. In *Proc. of the 32nd Conf. on Decision and Control, volume 3*, pages 2317 2322, 1993.

[Tay94] J. H. Taylor. A Modeling Language for Hybrid Systems. In *Proc. of the IEEE/IFAC Joint Symposium on Computer-Aided Control System Design*, pages 339 344, Tucson (AZ), USA, 1994.

[TB04] F. D. Torrisi and A. Bemporad. HYSDEL - A Tool for Generating Computational Hybrid Models for Analysis and Synthesis Problems. *IEEE Transactions on Control Systems Technology*, 12(2):235 249, 2004.

[TBB+02] F.D. Torrisi, A. Bemporad, G. Bertini, P. Hertach, D. HYSDEL - User Manual. Technical Report AUT02-10, ETHZ, Aug 2002.

[TBR01] E. Tetzner, R. Brauch, and G. Riedewald. Visual Modeling of Hybrid Systems for Symbolic Simulation in VYSMO. In *Modelling and Simulation 2001, 15th ESM'2001*, SCS Publication, 2001.

[TCB02] J.E. Tolsma, J. Clabaugh, and Paul I. Barton. ABACUSS II, Oct 2002. http://yoric.mit.edu/abacuss2/abacuss2.html, Massachusetts Institute of Technology.

[Tet00] E. Tetzner. Hybrid Systems for Symbolic Simulation of Time-dependent Control Systems. MOVEP'2k, June 2000. Nantes.

[Tet08] E. Tetzner. Wortbasierte Symbolische Simulation Hybrider Systeme in CLP. In M. Hanus and S. Fischer, editors, *Proc. of 25nd Workshop of GI-Fachgruppe 2.1.4 Programmiersprachen und Rechenkonzepte*, number 0811 in Technischer Bericht, pages 154 162. Christian-Albrechts-University Kiel, October 2008.

[TF00] L. Thévenon and J.-M. Flaus. Modular Representation of Complex Hybrid Systems: Application to the Simulation of Batch Processes. *Simulation Practice and Theory*, 8(5):283 306, 2000.

[TH00] H. Thae and D. Van Hung. Formal Design of Hybrid Control Systems: Duration Calculus Approach, 2000. Technical Report 221, UNU/IIST, P.O. Box 3058, Macau, November 2000.

[Tiw02] A. Tiwari. Formal Semantics and Analysis Methods for Simulink Stateflow Models, 2002. Technical Report, SRI International Menlo Park.

[Tiw03a] A. Tiwari. Approximate Reachability for Linear Systems. In *HSCC*, volume 2623 of *LNCS*, pages 514 525. Springer, 2003.

[Tiw03b] A. Tiwari. HybridSAL: Modeling and Abstracting Hybrid Systems, June 2003. SRI International Menlo Park, CA, http://citeseer.ist.psu.edu/695519.html.

[TKRE00] E. Tetzner, A. Kunert, G. Riedewald, and N. Erdmann. Symbolic Simulation for Cultivation of Biosensor Cells. In *FOODSIM2000*, SCS Publication, 2000.

[TLR04] E. Tetzner, W. Lohmann, and G. Riedewald. Rossy: A Tool for Symbolic Simulation of Hybrid Systems. In *Industrial Simulation Conference 2004*. EUROSIS, June 2004.

[TLR06] E. Tetzner, A. Lantsman, and G. Riedewald. Modellierung und Symbolische Simulation von Studiengängen am Beispiel des Grundstudiums der Informatik der Universität Rostock. Preprint CS-02-06, University Rostock, Department of Computer Science, 2006.

[TR99] E. Tetzner and R. Riedewald. MODEL-HS - An Approach to Specify Hybrid Systems for Symbolic Simulation. In A. Poetsch-Heffter and W. Goerigk, editors, *ATPS 99*, number 258 in Informatik-Berichte, pages 201 224. University Hagen, 1999.

[TR03] E. Tetzner and G. Riedewald. MODEL-HS: Translation into CLP for Symbolic Simulation, 2003. ISSN 0233-0784.

[TR05] E. Tetzner and R. Riedewald. Spezifikation und Verifikation in regelbasierten Beratungssystemen auf der Grundlage hybrider Automaten. In M. Hanus and F. Huch, editors, *Proc. of 22nd Workshop of GI-Fachgruppe 2.1.4 Programmiersprachen und Rechenkonzepte*, number 0513 in Technischer Bericht, pages 67 77. Christian-Albrechts-University Kiel, May 2005.

[TRB01] E. Tetzner, G. Riedewald, and R. Brauch. VYSMO and MODEL-HS - Userfriendly Specifications for Time- and Safety-Critical Systems. RIB, 2001. ISSN 0233-0784.

[TSR07] E. Tetzner, M. Sikos, and G. Riedewald. Automatic Transformation for Properties to Symbolic Simulation in Hybrid Systems. Computer Science Institut, Juni 2007.

[UR95] L. Urbina and G. Riedewald. A Framework for Symbolic Simulation of Hybrid Systems in Constraint Logic Programming. In *WLP*, pages 29 38, 1995.

[Urb96] L.A.J. Urbina. Analyse Hybrider Systeme in Constraint Logischer Programmierung. Dissertation, 1996.

[Var02] D. Varró. A Formal Semantics of UML Statecharts by Model Transition Systems. In *ICGT '02*, number 2505 in LNCS, pages 378 392, London, UK, 2002. Springer-Verlag.

[vBMR+06] D. A. van Beek, K. L. Man, M. A. Reniers, J. E. Rooda, and R. R. H. Schiffelers. Syntax and Consistent Equation Semantics of Hybrid Chi. *J. Log. Algebr. Program.*, 68(1-2):129 210, 2006.

[vBMR+07] D. A. van Beek, K. L. Man, M. A. Reniers, J. E. Rooda, and R. R. H. Schiffelers. Relating Hybrid Chi to other Formalisms. *Accepted for publication in: Electronic Notes in Theoretical Computer Science*, 2007.

[vBR00] D. A. van Beek and J. E. Rooda. Languages and Applications in Hybrid Modelling and Simulation: Positioning of Chi. *Control Engineering Practice*, 8(1):81 91, 2000.

[vdB00] M. von der Beeck. A Concise Compositional Statecharts Semantics Definition. In *FORTE'00*, volume 183 of *IFIP Conference Proceedings*, pages 335 350. Kluwer, 2000.

[vdSS00] A.J. van der Schaft and J.M. Schumacher. *Introduction to Hybrid Dynamical Systems*. Springer-Verlag, London, UK, 2000.

[Ver95] J. J. Vereijken. A Process Algebra for Hybrid Systems. In Bouajjani and Maler, editors, *The Second European Workshop on Real-Time and Hybrid Systems*, Grenoble, France, 1995.

[VPV06] E. Villani, E.M. Paulo, and R. Valette. *Modelling and Analysis of Hybrid Supervisory Systems: A Petri Net Approach (Advances in Industrial Control)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

[Wan07] F. Wang. Symbolic Simulation-Checking of Dense-Time Automata. In *FORMATS*, volume 4763 of *LNCS*, pages 352 368. Springer, 2007.

[Web97] M. Weber. *Systematic Design of Embedded Control Systems - Composing Models of System Structure and Behaviour*. Oldenburg Verlag, New York, 1997.

[WFSE96] K. Wöllhaf, M. Fritz, C. Schulz, and S. Engell. BaSiP - Batch process simulation with dynamically reconfigured process dynamics. *Proc. of ESCAPE-6, Computation and Chemical Engineering*, 972(20):1281 1286, 1996.

[WH06] D.K. Wittenberg and T.J. Hickey. Modeling Hysteresis in CLIP - The Tank Flow Problem. In *Proc. of the NFS Workshop on Reliable Engineering Computing*, Savannah, GA, 2006.

[Wie03] R.J. Wieringa. *Design Methods for Reactive Systems: Yourdon, Statemate and the UML*. Morgan Kaufmann Publishers, 2003.

[Wit04] D.K. Wittenberg. *CLP(F) Modeling of Hybrid Systems*. PhD thesis, The Faculty of the Graduate School of Arts and Sciences, Brandeis University, Department of Computer Science, 2004.

[WS95] R. Wieting and M. Sonnenschein. Extending High-level Petri Nets for Modeling Hybrid Systems. *In: A. Sydow, Systems Analysis Modeling Simulation, Gordon and Breach*, 18 19:259 262, 1995.

[WW99] S. A. Wolfman and D. S. Weld. The LPSAT Engine & Its Application to Resource Planning. In *IJCAI '99: Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence*, pages 310 317, San Francisco, CA, USA, 1999. Morgan Kaufmann Publishers Inc.

[WZP03] B. Woźna, A. Zbrzezny, and W. Penczek. Checking Reachability Properties for Timed Automata via SAT. *Fundamenta Informaticae*, 55(2):223 241, 2003.

[YLWD07] W. L. Yeung, K. R. P. H. Leung, Ji Wang, and Wei Dong. Modelling and model checking suspendible business processes via statechart diagrams and CSP. *Science Computer Programming*, 65(1):14 29, 2007.

[Yov97] Sergio Yovine. Kronos: A Verification Tool for Real-Time Systems. *International Journal of Software Tools for Technology Transfer*, 1, 1997.

[ZHL95] C. Zhou, D. V. Hung, and X. Li. A Duration Calculus with Infinite Intervals. In *FCT'95*, volume 965 of *LNCS*, pages 16 41, 1995.

[Zho93] C. Zhou. Duration Calculi:An Overview. In *Proc. of Formal Methods in Programming and Their Applications*, volume 735 of *LNCS*, pages 256 266, 1993.

[ZPK00] B. P. Zeigler, H. Praehofer, and T. G. Kim. *Theory of Modeling and Simulation*. Academic Press, January 2000.

[ZSKP95] Bernard P. Zeigler, Hae Sang Song, Tag Gon Kim, and Herbert Praehofer. DEVS Framework for Modelling, Simulation, Analysis, and Design of Hybrid Systems. In *Hybrid Systems II*, pages 529 551, London, UK, 1995. Springer-Verlag.

Index

Thesen

These 1 Die Sprachen MODEL-HS und VYSMO, die in der vorliegenden Arbeit entwickelt wurden, bieten erstmalig eine deklarative und modulare Beschreibung hybrider Systeme speziell zur symbolischen Simulation auf Basis formaler Sprachen in CLP.

These 2 Zur Schaffung der Sprachen sind besondere Merkmale der symbolischen Simulation auf Basis formaler Sprachen in CLP als einmalig entwickelter Ansatz analysiert.

These 3 Als textuelle und graphische Notationen sind MODEL-HS und VYSMO problemadäquate und nutzerfreundliche Beschreibungssprachen.

These 4 Mit der Entwicklung der Sprachen wird die Lücke zwischen der schwierigen Lesbarkeit bzw. Wartung komplexer CLP - Beschreibungen und problemadäquater sowie nutzerfreundlicher Beschreibungen geschlossen.

These 5 MODEL-HS und VYSMO basieren auf derselben formalen Grundlage, die aus hierarchisch und synchronisierend hybriden Automaten (HHA und SHA) besteht.

These 6 MODEL-HS und VYSMO stellen eine deklarative, problemorientierte Syntax und Semantik, ein angepasstes Typkonzept und modulare Strukturen zur Wiederverwendung im Sinn der Hierarchisierung und Synchronisation zur Verfügung.

These 7 Während MODEL-HS eng an SDL angelehnt ist, basiert VYSMO auf der Idee kommunizierender hierarchischer Zustandsautomaten.

These 8 Hybride Systeme werden auf der Basis hierarchisch und synchronisierend hybrider Automaten zur symbolischen Simulation, die einen akzeptierenden Prozess von Wörtern unter zeitlicher Betrachtung darstellen, beschrieben.

These 9 Symbole von Zeitwörtern hybrider Systeme werden mit Bedingungen kontinuierlicher und diskreter Komponenten in Abhängigkeit sich kontinuierlich ändernder Fortschrittsfunktionen physikalischer Größen verbunden.

These 10 Transitionsbeschreibungen und Synchronisationsverbindungen lassen die Bildung von Zeitwörtern formaler Sprachen zu.

These 11 Die Akzeptanz von Zeitwörtern als zentrales Thema der symbolischen Simulation wird im Zusammenhang mit der Hierarchisierung und Synchronisation hybrider Systeme neu definiert.

These 12 Das wohldefinierte Akzeptanzverhalten, welches in dieser Arbeit eingeführt wird, legt eine Strategie zur Hierarchisierung von hybriden Automaten fest.

These 13 Eigenschaften zum wohldefinierten Akzeptanzverhalten lassen sich in flachen Automaten, die den HHAs und SHAs semantisch zugrunde liegen, korrekt nachweisen.

These 14 Wohldefinierte Schnittstellen und lokale Deklarationsteile hierarchisch und synchronisierend hybrider Automaten weisen Parameter und Variablen in Bezug auf die physikalische Umwelt der beschriebenen Systeme und deren Programmsteuerung auf, die festgelegten syntaktischen und semantischen Voraussetzungen zur Hierarchisierung und Synchronisation entsprechen.

These 15 Hybride Systeme lassen sich zur effizienten Ausführung der symbolischen Simulation bequem nach CLP übersetzen.

These 16 Die symbolische Simulation, welche dem Beweis der Korrektheit hybrider Systeme bezüglich zeit- und sicherheitskritischer Eigenschaften dient und in der Art einer Simulation ausgeführt wird, ist eng mit dem Bounded Model Checking verbunden.

These 17 Durch die Verbindung zum Bounded Model Checking lassen sich Erkenntnisse aus dem Bereich der Analyse hybrider Systeme in den Bereich der formalen Sprachen integrieren bzw. Rückschlüsse aus dem Bereich der formalen Sprachen für den Bereich der Analyse hybrider Systeme ziehen.

These 18 In zukünftigen Arbeiten kann CLP zur Ausführung der symbolischen Simulation durch Erkenntnisse und Erfahrungen aus dem Bereich des SAT-basierten Bounded Model Checking erweitert werden.

These 19 Die Komplexität zur Transformation und Ausführung hybrider Systeme in CLP kann durch die Nutzung geschachtelter Wörter verbessert werden.

These 20 Zur Beschreibung hybrider Systeme und deren Ausführung mit Hilfe der symbolischen Simulation existiert das Werkzeug ROSSY, welches vollständig konzipiert ist und dessen Implementationsplattform Prolog IV darstellt.

These 21 Anfragen nach zeit- und sicherheitskritischen Eigenschaften können in Form von temporal-logischen Ausdrücken intuitiv entsprechend Erfahrungen aus dem Bereich des Model Checking bzw. durch Anfragemasken für Programmierer praktischer Anwendungsgebiete nutzerfreundlich und anwendungsnah beschrieben werden.

These 22 Die Anwendung von MODEL-HS und VYSMO in der Modellierung von Studiensystemen und parallelen Programmen liefert neue Erkenntnisse zur Analyse verschiedener Systeme von Bedingungen sowie zeit- und sicherheitskritischer Eigenschaften und belegt die komfortable Beschreibung mit hybriden Systemen in zeit- und sicherheitskritischen Bereichen.